Polyglot software development

POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria informatica e dei sistemi – XXVI ciclo

Tesi di Dottorato

Polyglot software development

Benefits, problems and guidelines for the adoption and combination of domain specific formal languages

Federico Cesare Argentino Tomassetti

Tutore
prof. Marco Torchiano

Coordinatore del corso di dottorato
prof. Pietro Laface

December 2013

Summary

The languages we choose to design solutions influence the way we think about the problem, the words we use in discussing it with colleagues, the processes we adopt in developing the software which should solve that problem.

Therefore we should strive to use the best language possible for depicting each facet of the system. To do that we have to solve two challenges: i) first of all to understand merits and issues brought by the languages we could adopt and their long reaching effects on the organizations, ii) combine them wisely, trying to reduce the overhead due to their assembling.

In the first part of this dissertation we study the adoption of modeling and domain specific languages. On the basis of an industrial survey we individuate a list of benefits attainable through these languages, how frequently they can be reached and which techniques permit to improve the chances to obtain a particular benefit. In the same way we study also the common problems which either prevent or hinder the adoption of these languages. We then analyze the processes through which these languages are employed, studying the relative frequency of the usage of the different techniques and the factors influencing it. Finally we present two case-studies performed in a small and in a very large company, with the intent of presenting the peculiarities of the adoption in different contexts.

As consequence of adopting specialized languages, many of them have to be employed to represent the complete solution. Therefore in the second part of the thesis we focus on the integration of these languages. Being this topic really new we performed preliminary studies to first understand the phenomenon, studying the different ways through which languages interact and their effects on defectivity. Later we present some prototypal solutions for i) the automatic spotting of cross-language relations, ii) the design of language integration tool support in language workbenches through the exploitation of common meta-metamodeling.

This thesis wants to offer a contribution towards the productive adoption of multiple, specific languages in the same software development project, hence polyglot software development. From this approach we should be able to reduce the complexity due to misrepresentation of solutions, offer a better facilities to think about problems and, finally to be able to solve more difficult problems with our limited

brain resources.

Our results consists in a better understanding of MDD and DSLs adoption in companies. From that we can derive guidelines for practitioners, lesson learned for deploying in companies, depending on the size of the company, and implications for other actors involved in the process: company management and universities.

Regarding cross-language relations our contribution is an initial definition of the problem, supported by some empirical evidence to sustain its importance. The solutions we propose are not yet mature but we believe that from them future work can stem.

Acknowledgements

Dear hyphotethical reader, I am sure you will understand my reasons to write most of this part of the thesis in my native language, considering it is mainly addressed to other Italian speakers. However, in accordance to the title, a few other languages are used in these pages.

Ho imparato quanto sia vero che gli esami non finiscono mai. Non mi illudo di poterli considerare quindi del tutto terminati, cionondimeno con questa tesi e, sperabilmente, con un esito positivo degli esami di dottorato, raggiungo un traguardo che merita una piccola celebrazione. Per quanto questo percorso sia stato lungo e irto di ostacoli lo sarebbe stato ancora di più (molto, anzi infinitamente di più) senza il supporto, in forme diverse e complementari, di una miriade di soggetti.

Vorrei prima di tutto ringraziare il mio relatore, il prof. **Marco Torchiano** per avermi guidato in questo percorso diversi anni orsono, iniziato con la tesi di laurea specialistica. Ero allora ben determinato a **non** intraprendere questa strada, tuttalpiù a spendere una manciata di mesi in dipartimento e poi...

Ed eccomi qua, immune a ogni forma di coerenza.

Sebbene certi momenti sia stato leggermente meno che entusiasta di aver fatto questa scelta, a posteriori sono felice di averci ripensato e gli sono quindi grato di avermi proposto questa apparentemente malsana idea.

Un grazie lo devo anche al prof. **Maurizio Morisio**, con cui avevo svolto la tesina di laurea triennale (sebbene lui non se ne ricordi). Ammiro molto la sua capacità di cogliere in un attimo il nocciolo del problema. Al contempo sono un po' spaventato dalla sua innata abilità nello scovare la magagna che credi di aver ben nascosta.

Ora, se Newton ha dichiarato di aver potuto vedere più in là solamente perché abbarbicato sulle spalle di giganti, io, per ogni contributo anche minimo che possa aver offerto alla ricerca in questi anni, ho dovuto appoggiarmi a una stirpe di titani di proporzioni biblico-apocalittiche.

Fra questi mi fa piacere ricordare **Antonio Vetrò**, un uomo perennamente in lotta con le codifiche di carattere e le citazioni che non gli vengono riconosciute. Con lui ho orgogliosamente costituito la trasposizione nel mondo della ricerca di grandi

coppie che hanno segnato l'immaginario collettivo. Ricordateci così, come i *Gianni e Pinotto* della ricerca italiana.

Appena ho potuto ho cercato scuse per lavorare con **Giuseppe Rizzo** perché è molto, molto bravo. Lavorare con lui è stato sempre stimolante. Non ho capito tutto quello che ha provato a insegnarmi, in compenso ho annuito moltissimo.

Con **Luca Ardito** ho condiviso lo stesso percorso e si è cazzeggiato il giusto. Bene così.

Ho incrociato tante altre anime più o meno pie in questi laboratori dei piani bassi. Li ricordo tutti con affetto: dai compagni di gruppo (**Oscar**, **Cristhian**, **Alì** e **Najeeb**) ai compagni di laboratorio (**Andrea Martina**, mio fiero compagno di scrivania per anni, **Matteo** e **Sham**).

Ringrazio **Sara** per i molte caffè, pranzi, sedute di psicoterapia fra dottorandi.

Ci sono stati altri ricercatori più esperti da cui ho potuto imparare molto (o almeno, loro me ne hanno offerto la possibilità). Fra questi **Filippo Ricca**, che ha anche scritto per me una lettera di raccomandazione che ha contribuito a farmi ottenere la borsa DAAD grazie alla quale ho speso un periodo in Germania. Del gruppo di Genova ringrazio anche **Alessandro Tiso**, **Gianna Reggio** e **Maurizio Leotta**.

In Germany I had the possibility to work with **Daniel Ratiu** and **Bernhard Schätz**.

Daniel gave me a lot of practical advices (*Put a figure in the first page!*) and shared with me its wisdom. So I learnt to *îngrășa porcul înainte de Crăciun*.

There I also met one of my idol: **Markus Völter**. To have the possibility to discuss with him, **Bernd Kolb** and **Domenik Pavletic** were very good reasons to bear a winter in München.

Credo che senza il contributo di chi lavora sul campo, e produce software veramente, molte idee non sarebbero mai nate e avrei spesso imboccato sentieri (ancora più) improbabili. Fortunatamente abbiamo avuto la possibilità di scrivere alcuni lavori con Trim, di cui ringrazio **Lorenzo Bazzani** e **Felice Di Luca**, e CSI Piemonte, di cui ringrazio **Mauro Antonaci** e **Paolo Arvati**.

There is a full list of people who gave opinions, took part in experiments, surveys, interviews. There were students who shared ideas with me, who worked on bachelor and master thesis in directions helpful for my research. I could keep listing names for pages, let me instead sum up in a big

THANK YOU, GRAZIE, DANKE, GRACIAS

E i compagni di corso di dottorato, e i professori. Per voi ci sono parole che risuonano, con voce stonata, da qualche tempo:

Vivat academia!
Vivant professores!
Vivat membrum quodlibet;
Vivant membra quaelibet;
Semper sint in flore.

E poi c'è il resto, l'esistenza, la Vita, il tutto. In ordine rigorosamente sparso.

Luca Barbato, che conosco dal principio, mi ha dato diverse buone idee, molta cioccolata e supporto attraverso quest'avventura.

C'è la tesi di **Luca Gilli** che è stato un esempio. Da leggere, sfogliare e capirne l'introduzione. Poi, quando il gioco si fa serio e la matematica sovrabbondante, mi ritiro in buon ordine.

There is **Shalini**. By looking her near to my old friend I feel all of a sudden the distance more bearable. Keep him in good hands. *Chala Bagundi!*

La Goliardia, perché rinverdisce l'amore per l'Università e quello che rappresenta. Al di là delle momentanee difficoltà o errori (*ad augusta per angusta*). Perché entrambe siete eterne.

I nostri vecchi chi hanno insegnato a dividere il pane ed il sale e ci hanno dato le gambe per incamminarci. E il cuore. Oggi ci mandano il vaglia delle loro trepidazioni.

Bobo Rossi, **Un Goliardo**

Vorrei ringraziare i nonni di cui porto il nome e a cui ripenso nei momenti di difficoltà. Non ho avuto l'onore di conoscerli, però il loro esempio risuona nelle persone che gli sono state accanto, nei gesti e in ciò che è rimasto dopo di loro. Perciò sono grato a mio nonno **Argentino** che ha trovato la forza per venire a Torino, mantenersi durante gli studi e laurearsi in questo ateneo nel 1929. Un grazie a mio nonno **Cesare**, che non potè laurearsi ma fu capace di diventare impresario e progettare edifici e strade, in Italia e all'estero. Le foto, i racconti e il tuo tavolo mi sono stati di monito fin dall'infanzia.

A mia madre per il supporto economico durante gli studi.

A mia zia **Antonietta** che mi ha sempre incoraggiato e che mi chiama Ingegnere da tempo immemore. Senza il suo supporto in termini d'affetto e pratico tutto sarebbe stato più difficile.

Mia zia **Giuliana** e mio zio **Giorgio** mi hanno spronato anch'essi. Quando andai a lavorare dopo aver conseguito la laurea triennale fui minacciato di essere preso a calci nel sedere. Non rinsavii subito ma col tempo ragionai su quelle parole e sono tornato al Poli. Un po' più a lungo del previsto.

A mia zia **Elisabetta** devo molto, mi spiace di non aver saputo ricambiarla dell'affetto e della sua generosità.

A special thank you goes to marvellous, astonishing people I met during the periods I spent abroad. You made my life a lot more interesting.

Obrigado **Diogo**, você foi o melhor amigo de Erasmus que eu jamais poderia ter.

Ti, **Mojca**, **Domen** in Slovenija imate mesto v mojem srcu. Tudi brez jezera.

Flavia, când sunt trist îmi amintesc de modul tău de a zâmbi și redevin fericit. Aștept să ne revedem, cine știe cand și unde.

Victor, because, man, it was a lot of fun!

Daniel, muito grato por todo o Porto. Tudo de bom para o seu PhD!

Marcé, le risate che ci hai fatto fare e la tua carica mi fanno venir voglia di essere migliore di quello che sono.

Giovanna, per i bei momenti in Veneto, Germania e Spagna. Al prossimo Spritz!

Giulia S., aver vissuto vicino a te a Monaco mi ha fatto sentire bene. Mi manca avere una vicina bella, simpatica e solare come te. Lo sai che devi venirmi a trovare, vero?

Claudio e **Daniele**, pezzi di cuore espatriati in Germania. Claudio, ricordati che dovunque tu vada corri sempre il rischio di ritrovarti con una Z sulla fronte e dover fornire spiegazioni alle autorità competenti. E in tedesco, la vedo dura. Anche se quella volta a St. Louis...

Andrea e **Giulia N.**, i coinquilini che hanno alliettato l'ultimo anno. O che mi hanno dato un sacco di buone e di pessime idee negli ultimi quindic'anni.

Camille, perchè a volte i francesi non sono persone orribili. *Merci beaucoup mon chou.*

Ringrazio mia nonna **Ivonne**, che è mancata mentre scrivevo le ultime pagine. Comunicavamo con poche parole, io e te, quelle giuste sai dove puoi leggerle. Grazie di tutto, nonna.

Per ultimo ringrazio mio Padre, l'alpha e l'omega dei miei pensieri, perché non c'è cosa che faccia o provi a fare, in cui non senta la tua presenza. Scusami se non sono riuscito a fare di più

Federico Cesare Argentino Tomassetti

List of Tables

List of Figures

Contents

Chapter 1

Introduction

> *I observe a cultural tradition, which in all probability has its roots in the Renaissance, to ignore this influence, to regard the human mind as the supreme and autonomous master of its artifacts. But if I start to analyze the thinking habits of myself and of my fellow human beings, I come, whether I like it or not, to a completely different conclusion, viz. that the tools we are trying to use and the language or notation we are using to express or record our thoughts are the major factors determining what we can think or express at all! The analysis of the influence that programming languages have on the thinking habits of their users, and the recognition that, by now, brain-power is by far our scarcest resource, these together give us a new collection of yardsticks for comparing the relative merits of various programming languages [Dijkstra, 1972].*

The languages we use to represent, formalize and think about our programs have long reaching effects on the way we look at problems, envision the solutions and the processes we adopt to bridge the problems to our solutions; languages are the most important tools we use to create software. This thesis address this broad research questions:

How can we adopt and combine specific languages for software development?

In this Chapter we motivate the importance of language, notations and their supporting tools discussing their effects on all mental activities (Sect. 1.1), and in particular on the mental activity we are more interested into: programming (Sect. 1.2). Later we briefly summarize software development paradigms designed to higher the level of abstractions (Sect. 1.3). Finally we present the research design we adopted to address the goal stated few lines above (Sect. 1.4).

1.1 External factors affecting our ability to think

As human beings the ability to use tools to improve our capacities is arguably among our most important skills. As a species, we have been able to improve our natural performance and our chances of accessing the food, protect ourself by predators and improve our living conditions. Thanks to tools human beings managed to emerge among stronger and faster competitors.

Nowadays the most common challenges at hand for human beings are not anymore physical but are instead typically intellectual. Still we can benefits from tool to support also these activities.

> *Just as you cannot do very much carpentry with your bare hands, there is not much thinking you can do with your bare brain (quoted in [Dennett, 1996])*

This sentence expresses the commonality between physical and mental activities. It is commonly known that tools augmented the physical possibilities of human beings. On the same way there is a category of tools leveraging the human mental potential.

To understand the effect of the external factors considered (language, notation and tools) used on our ability to think let's consider some examples related to mathematics. Many operations are simply not possible without the concept of zero (language), multiplications are more difficult to execute using roman numbers in respect to arabic numbers (notation). Finally consider the kind of operations you are able to solve with or without using pen and paper (tools). Let's consider all of these aspects in more details.

1.1.1 Languages

The origin of language has been for decades one of the most debated topic in the scientific community. Different researchers suggest that the role of the language in developing our mental abilities as a species was fundamental. We report the opinion of Charles Darwin:

> *The mental powers in some early progenitor of man must have been more highly developed than in any existing ape, before even the most imperfect form of speech could have come into use; but we may confidently believe that the continued use and advancement of this power would have reacted on the mind by enabling and encouraging it to carry on long trains of thought. A long and complex train of thought can no more be carried on without the aid of words, whether spoken or silent, than a long calculation without the use of figures or algebra [Darwin, 1871, p.57].*

Logan suggests that languages emerge to cope with overwhelming complexity [Logan, 2007]. The first language would have been emerged to permit to move reasoning from perceptions of single facts to the manipulation of concepts, grouping similar facts under one single label. According to Logan writing can be regarded as a new language, different from speech. It would permit to order complexity in a more structured way, than it is possible with oral natural languages. This consideration is supported by different findings on structural and semantic differences between speech and writing [Akinnaso, 1982]. Language is not just a mean to communicate but it is also a mean for reasoning and thinking: different forms of language (oral and writing) support differently these activities.

Dennett suggests that languages are the tools permitting to develop explicit generalization, while animals who can not use proper languages are limited to implicit generalization [Dennett, 2000]. In this sense the language used becomes a tool enabling more powerful form of abstractions.

In the first half of the twentieth century the idea of strong linguistic relativity was supported with enthusiasm by the majority of the academical community. According to this principle the language we speak determine absolutely what we are able to think about. This principle was derived from the initiated by Sapir and prosecuted by Whorf and it is also referred as the Sapir-Whorf hyphothesis. Here follows some extracts:

> *Human beings do not live in the objective world alone, nor alone in the world of social activity as ordinarily understood, but are very much at the mercy of the particular language which has become the medium of expression for their society. It is quite an illusion to imagine that one adjusts to reality essentially without the use of language and that language is merely an incidental means of solving specific problems of communication or reflection. The fact of the matter is that the 'real world' is to a large extent unconsciously built upon the language habits of the group. No two languages are ever sufficiently similar to be considered as representing the same social reality. The worlds in which different societies live are distinct worlds, not merely the same world with different labels attached... We see and hear and otherwise experience very largely as we do because the language habits of our community predispose certain choices of interpretation. [Sapir, 1929, p. 69]*

> *We dissect nature along lines laid down by our native languages. The categories and types that we isolate from the world of phenomena we do not find there because they stare every observer in the face; on the contrary, the world is presented in a kaleidoscopic flux of impressions which has to be organized by our mindsand this means largely by the linguistic*

> *systems in our minds. We cut nature up, organize it into concepts, and ascribe significances as we do, largely because we are parties to an agreement to organize it in this wayan agreement that holds throughout our speech community and is codified in the patterns of our language. The agreement is, of course, an implicit and unstated one, but its terms are absolutely obligatory; we cannot talk at all except by subscribing to the organization and classification of data which the agreement decrees.[Whorf, 1940, pp. 21314]*

The idea that the language could determine in an absolute manner what human beings are able to think was then criticized and rejected from the scientific community. Recently the idea returned in a relaxed form: different experiments now confirm an influence of the language used on the way of thinking (e.g., [Gumperz and Levinson, 1996, Pütz and Verspoor, 2000]) while admitting the possibility of thinking concepts for which a word does not yet exist. In that case we are lacking support for *manipulate* that concept, and related reasoning would be more difficult but not strictly impossible.

1.1.2 Notations

In the field of mathematics the usage of improved notations made possible for a vast majority of the population to develop skills that were limited to the brightest minds of the past. Whitehead explains that in details:

> *By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that ... a large proportion of the population of Western Europe could perform the operation of division for the largest numbers. This fact would have seemed to him a sheer impossibility ... Our modern power of easy reckoning with decimal fractions is the almost miraculous result of the gradual discovery of a perfect notation. [...] By the aid of symbolism, we can make transitions in reasoning almost mechanically, by the eye, which otherwise would call into play the higher faculties of the brain. [...] It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilisation advances by extending the number of important operations which we can perform without thinking about them. Operations*

> *of thought are like cavalry charges in a battle – they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.[Whitehead, 1911, p. 59]*

While the introduction of arabic ciphers into the western civilization contribute to bring in the reach of human minds progresses which were simply unthinkable before there is still room for improvements. Natural languages which employs simpler schemas to express numbers lead to higher performance in international mathematics competition. That is the case for example of the Korean language represents numbers. In Korean numbers from 1 to 10 are represented by words of just one syllable and the system is consistently regular. The effects on that on calculation performance were studied in depth [Park, 2004].

1.1.3 Tools

Andy Clark et al. introduced the concept of *active externalism*. Considering the active role of the environment in driving cognitive processes they suggest that the human brain and the external tools are part of a complexive system, i.e. tools used to support reasoning can be seen as an extension of the mind [Clark and Chalmers, 1998].

Clark suggests that the biological brain has evolved and matured in ways which factor in the reliable presence of a manipulable external environment. Nersessian concludes that engineering scientists think by means of the artifact models they design and build and without them they are almost unable to think [Nersessian, 2009]. So tools can support designing and thinking, and so also programming which is a specific mental activity. In some way languages can be seen as particular tools supporting reasoning.

Technology in particular made available an unprecedented set of tools that we can exploit to improve the possibilities of out mind. According to some authors the border itself from our proper mind and the tools it is taking advantage of, is difficult to establish, it sometimes blurry, making in some sense us a sort of cyborgs:

> *What used to be tools and machines that we could keep at arms length, has crept up on us, turning into something with which we constantly interact. People and technology have become intertwined. You cannot understand the one without understanding the other.[Dahlbom, 1996]*

1.2 External factors affecting our programming ability

In the previous section we examined how languages, notations and tools affect in general the ability to execute mental tasks. In this section we present some findings about the influence of the same factors on the set of mental tasks related to software development and in particular to programming.

1.2.1 Languages

Paul Graham introduces the Blub Paradox [Graham, 2001]: by interpreting programming through the lens of the programming languages we know, we are not able to understand the benefits and power of other paradigms. Developers knowing only "less powerful" languages are unable to appreciate possibilities given by "more powerful" languages because they are not able to consider the concepts that more powerful languages permit to express. We tend to interpret this particular mental activity according to our main tool of the trade: the programming language that we are using. As consequence, by adopting more "powerful" languages, we acquire more ways to think about software, we get new prespectives on it which could make solving a problem significantly easier. The definition of "powerful" is not strict and we believe it is relative to the kind of problems considered. In a certain case it could be a better support for regular expressions, in other cases an advanced modularity or the support to functions as first-citizens. However there are studies that were conducted on approached to measure the "power" of languages in general [Felleisen, 1990, Mitchell, 1993]. Other studies were conducted on the power of specific languages [Hidders et al., 2005].

Some comparisons were made between programming languages and other formal languages. For example Libkin [Libkin, 2001] compared SQL and relational algebra. It results that SQL cannot express recursive queries, which are necessary to determine reachability. The author proved that recursion (introduced in SQL3) add to the expressive power of the previous version of the language (SQL2) because it enriches the possibility to express concepts which were not expressable before. In our opinion the power of a language is not determined just by what it permits or permits not to express something but also how naturally it permits to express it: direct, not redundant support for concepts relief the brain from unnecessary work that has to be done when the language permit to express something, but only in a redundant, obscure way.

However some studies were conducted to evaluate a-posteriori the power of a language or to confront were devised, still into the foreseeable future defining new abstractions remains a conquer for the human mind.

1.2.2 Notations

The notation is the representation through which the developer perceive the language. The most common notations in software development are textual but alternatives are possible: graphical notations (e.g., UML) or spreadsheet notations (e.g., Excel). Tabular notations were also experimented, for example to express binary formulas [Pane and Myers, 2000].

In 1997 Whitley [Whitley, 1997] organized empirical evidence for and against graphical notations. Graphical notations help to provide better organization to the information displayed and make explicit relations which are normally implicit in textual notations (for example, using shared identifiers). Graphical notations tend to be more useful as the size or complexity of the problem grows [Day, 1988]. Moreover they are particularly appreciated by non-programmers [Baroth and Hartsough, 1995], who benefit more from them [Neary and Woodward, 2002]. Graphical notations were employed in Scratch: an IDE tailored for teaching programming at children between 8 and 16 years old [Maloney et al., 2010]. A similar project is Alice [Dann et al., 2008]. One reason for the better performance with graphical notations could be that they lead to build in-mind better models of the problem [Navarro-Prieto and Cañas, 2001]. One problem of graphical notation is the Deutsch Limit[1] (the lower screen density of graphical notations in respect to textual notations). We wonder if the simple fact that larger and cheaper monitors are available on the market could slightly reduce the importance of this aspect.

It emerges that every evaluation depends on the task being performed [Green, 1989]: a certain notation can be the most suitable for a particular task but terrible for another, leading to poorer performance in respect to both time of completion and correctness. For some particular tasks some comparisons between different notations were made, for example by Andrews [Andrews and Schneider, 1983] on comprehension and analysis of concurrent programs.

A correct notation also improves our ability to reason on a particular model. Gross [Gross, 2009] in particular discuss the role of graphical notation.

Iverson discusses about characteristics that should be embodied by a good notation in his Turing Award lecture [Iverson, 1980] and presented some arguments on the usefulness of multiple notations. He discusses how these characteristics can influence our way to write programs, namely: executability, universality, ease of expressing constructs arising in problems, suggestivity, ability to subordinate detail, economy, amenability to formal proofs.

While arguably, the most used programming language is Excel, with its spreadsheet paradigm more traditional software development still is based almost solely on

[1] `http://en.wikipedia.org/wiki/Deutsch_limit`

textual notations. This is due to different reasons, we wonder if all of these reasons are intrinsic in the difference between these notations or if they could depend on a superior tool support developed for textual notations for diff/comparing, storing, etc. Merkle [Merkle, 2010] explained the advantages of the textual notation for DSLs but we think the same apply to all languages, also GPLs. Among the advantages for textual notation he listed: easy information exchange via e.g. mail, integration into existing tools like diff, merge and version control and most important the fast editing style supported by the "usual" IDE support like code completion, error markers, intentions and quick fixes. Some development activities are already largely performed using visual notations, for example GUI builders permit to define the layout of widgets in a visual way, with the corresponding code being generated later. Maybe more appropriate tool support could enable the adoption of different notations.

Considering that more notations can be attached to the same language the optimal solution could be the adoption of different notations to perform different tasks, optimizing the performance achieved in each of these tasks, and letting developers gain different insights by looking at the same code in different forms.

1.2.3 Tools

Kasurinen et al. [Kasurinen et al., 2013] analyzed the role of tools supporting a particular kind of software development: game development. They reported that the quality of tools is perceived as high. Practitioners value at most the flexibility of tools, which permits to achieve adaptability to changing requirements and plans. We believe that the quality of supporting tools in this branch of software development permitted to improve greatly the productivity. The same quality of *domain-specific* supporting tools is not available in other branches and that is unfortunate.

Bolchini et al. [Bolchini et al., 2008] studied how hypermedia development of large applications. can be carried out by non-professional thanks to high quality of available development tools, which enables them to focus on the domain knowledge they have, instead of the technical knowledge required to develop traditionally the application.

Tony Clark and al. writing about Language-Driven Development [Clark et al., 2004] described how modeling tools can support reasoning:

> Rather than dealing with a plethora of low level technologies, developers can use powerful language abstractions and development environments that support their development processes. They can create models that are rich enough to permit analysis and simulation of system properties before completely generating the code for the system. They can manipulate their models and programs in significantly more sophisticated

> ways than they can code. Moreover, provided the language definitions are flexible, they can adapt their languages to meet their development needs with relative ease.[Clark et al., 2004, p. IX]

Effective tools can be developed to support specific manipulations of concepts that are idiomatic to a particular domain. To reach this goal we need higher-level, domain specific languages. The possibility to have advanced tools is enabled by the availability of more appropriate languages.

1.3 Programming paradigms based on improved abstractions

Programming languages initally were conceived to let programmers think in the terms of the machine hardware, so that they could write code as optimized as possible. The next important step was to create a sort of portable assembler (C); portability was the goal which made acceptable a small increment in the level of abstraction at the cost of a (slight) performance hit. As hardware improvements make sustainable to not focus exclusively on performance, programming evolved towards increasingly higher levels of abstactions. A few industry sectors were then able to sustain "their own" language: FORTRAN (1955) for engineering and scientific community, LISP (1958) for Artificial Intelligence applications, COBOL (1959) for business and financial services.

As programming gain importance new, more specialized languages were developed: from Pascal (1970) for teaching programming, ADA (1980) for development specific for the American department of defense, MATLAB (late '70) for development of numerical computing applications.

Over the years a number of very specialized languages, the so called Domain Specific Languages emerged to be used in substitution of General Purpose Languages for specific tasks. Among the other we cite LaTeX(1980), PostScript (1982), HTML (1991), CSS (1996).

An interesting attempt leap forward was Literate Programming, a term conceived by Donald Knuth:

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do. [Knuth, 1984]

Donald Knuth also realized the first tool supporting this approach. It was named WEB and it let describe the program in natural language embedding into these

description snippets of code. Later from the description a LaTeXdocument was generated and source code in Pascal was extracted.

The possibility to develop languages that fit the problem at hand, your mindset and your skills is an enormous advantage but higher abstrations can also be introduced by idioms and shared techniques for using a language. Consider Design Patterns[Gamma et al., 1994]: one of their goal is to build a jargon which experts in a certain field can use to communicate about their solutions and the way the achieve them. On Design Patterns there is interesting thought from Graham:

> This practice is not only common, but institutionalized. For example, in the OO world you hear a good deal about "patterns". I wonder if these patterns are not sometimes evidence of case (c), the human compiler, at work. When I see patterns in my programs, I consider it a sign of trouble. The shape of a program should reflect only the problem it needs to solve. Any other regularity in the code is a sign, to me at least, that I'm using abstractions that aren't powerful enough - often that I'm generating by hand the expansions of some macro that I need to write. [Graham, 2002]

On one hand Design Patterns permit to software designers to communicate, expressing in one word the path chose to create a solution to a problem, on the other hand the problem is solved only when communication from human to human is considered while the problem still remains in communication between human (the programmer) and the machine, i.e. the need of Design Patterns is a symptom that it is not possible to express to machine a certain concept which human can think as atomic, which human beings can abstract but the language permits not to.

In the rest of the section we present specific ways to build a specific vocabulary (Sect. 1.3.1) and your own abstractions (Sect. 1.3.2) possibly supported by your own syntax (Sect. 1.3.3).

1.3.1 Libraries development and Metaprogramming

A first method to higher the conceptual level of the code is to enlarge its dictionary through the development of new libraries, which permit to express concisely large and complex operations. These efforts are limited by the expressive power of the language. The concept of library of code appeared in 1959 [Wexelblat, 1981, p. 274] in the JOVIAL programming language. Another evolution step was the introduction of subprograms in FORTRAN. Nowadays the usage of standard libraries or external libraries is extremely common, with systems devoted just to maintain the libraries

used by each project: think about Maven for Java[2] or RubyGems[3] for Ruby.

Other way to enlarge the power of language is the adoption of metaprogramming techniques. Metaprogramming, which powers depends on the language, can permit to express concisely concepts which are later translated in automatic code modifactions. Some languages, in particular dynamic languages (e.g., Ruby) or functional languages (e.g., Lisp) have strong metaprogramming support which permits to change significantly the appearance of the languages. As consequence the concept of Internal Domain Specific Language was born.

1.3.2 Model-driven development

The idea of developing software based on models has been part of several approaches and methodologies proposed in the last 20 years [Booch, 1991]. Model-driven development (MDD) leverages detailed models by generating automatically the application code [Jiang and Hu, 2008] or interpreting them at runtime [Zeng et al., 2005]. Such approach has been proposed both for general purpose development [Jacobson et al., 1999] and for specific domains (e.g., real-time applications [Selic et al., 1994]).

The term "model" is very general; it is difficult to provide a comprehensive yet precise definition of a model. For us, a model is an artefact realized with the goal to capture and transfer information in a form as pure as possible limiting the syntax noise. Examples of models include UML design models [Object Management Group, 2011b], process models defined through BPMN [Object Management Group, 2011a], Web application models defined through WebML [Ceri et al., 2000] as well as textual Domain Specific Language (DSL) models[4] [Fowler and Parsons, 2011].

Given that models are so heterogeneous and the processes involving them so varied, it is actually difficult to define the exact boundaries for modelling. Moreover, models can be used practically in different ways and with different levels of maturity [Tomassetti et al., 2012]. While in theory we have a set of best practices, in practice we could find mixed, half baked and personalized solutions. As a consequence, the different uses of modelling are complex and difficult to classify precisely.

When models constitute a crucial and operational ingredient of software development then the process is called model driven. There are different model driven techniques: Model Driven Engineering (MDE) [Schmidt, 2006], Model Driven Development (MDD) [Mellor et al., 2003] and Model Driven Architecture (MDA) [Kleppe et al., 2003]. MDE is the broadest one; it is a software methodology that focuses

[2] `http://maven.apache.org/`

[3] `http://rubygems.org/`

[4] Many researchers would probaly not include DSLs among the model-driven techniques.

on creating and exploiting models, which are typically given as abstract representations using a modelling language (e.g., UML but also BPMN or a home-grown DSL [Fowler and Parsons, 2011]). Instead, MDD is a development approach that uses models as the main artefacts of the development process. In MDD, models at higher-level of abstraction are (progressively) transformed into models at lower-level of abstraction until the models are executable using either code generation or model interpretation. In this latter case, executable models are directly executed/interpreted by means of specific environments [Mellor and Balcer, 2002]. The main difference between MDE and MDD is that MDE goes beyond the pure development activities and encompasses also other software engineering tasks (e.g., models evolution and reverse engineering of legacy systems). Finally, MDA is a registered trademark of Object Management Group (OMG) recommending the usage of OMG standards (e.g., UML). MDA is more specific than MDD and defines the system functionality using the notions of platform-independent model (PIM) and platform-specific model (PSM). The main idea at the base of model-driven techniques is to realize higher level artifacts (i.e., *models*) from which implementation artifacts can be obtained either through code generation or through interpretation.

1.3.3 Language Specific Engineering

Domain Specific Languages are languages develop to express concisely a limited number of concepts pertaining to a specific concern. Domain Specific Languages can be used together with General Purpose Languages to create complete systems. However a few approaches were devised with the purpose of creating systems using a combined set of DSLs. We call the approaches based on the combination of specific languages as *Language Specific Engineering*.

For example Language Oriented Programming (LOP) is a term introduced by Ward in its paper in 1994 [Ward, 1994] This approach require the development of DSLs to represent each part of the system. The application is then composed by domain knowledge, expressed through DSLs and the generators able to translate this domain knowledge in the implementation.

Dimitriev discusses on LOP presenting the supporting workbench that was developed at JetBrains named Meta-Programming System (MPS) [Dmitriev, 2004]. He points out the need of a workbench that make development of new languages and supporting environement nearly as easy as common programming: develop a new language should be comparable to develop a new library but with the advantages brought by a specific syntax and specific tools. The goal is to make feasible for programmers to craft new tools to be used to design and implement complex systems instead of using general-purpose tools.

General-purpose languages require me to translate high-level domain

> concepts into low-level programming features, most of the big picture is lost in the resulting program. When I come back to the program later, I have to reverse engineer the program to understand what I originally intended, and what the model in my head was. Basically, I must mentally reconstruct the information that was lost in the original translation to the general-purpose programming language.[5]

The idea of LOP presented by Dimitriev require to abandon text form for storing programs.

Humm et al. introduced DSL stacking to realize Language Oriented programming [Humm and Engelschall, 2010]. This method for implementing Language-Oriented Programming develop DSLs and general-purpose languages incrementally on top of a base language.

Language engineering or metaprogramming aims to make these advantages available even to smaller projects, to make it feasible and convenient to develop languaages on a per-project basis. The problems to solve are many: first of all we have better LW to develop languages and supporting tools fast but we need good interoperability and good learnability, also guidelines and expertise in developing languages (until now it was a very rare skillset to be a language developer).

1.4 Research design

When facing the challenge posed by programming or in general develop models and systems we should choose wisely the best tools for the daunting tasks, hence choosing the right languages and the right notations for the task at hand is a fundamental part of the process.

Of course choose or even create the best language for the task pose a lot of different challenges. Languages have to be designed, evaluated and evolved. A language to be truly useful need an ecosystem of supporting tools: from compilers and editors to debuggers and testing facilities. Once the single elements of the system are realized with the most appropriate language, using the corresponding supporting tools those elements still need to be integrated in a coherent, cohesive system. The developers and the software engineers have the orchestrate those language and make them cooperate. These challenges are at the very core of the software engineering. This humble thesis hopes to offer a contribution in the direction of guiding the adoption of high level languages and combine them.

This thesis is divided in two parts: in the first one we analyze the adoption of modeling and domain specific languages, while in the second we focus on the

[5] See `http://www.onboard.jetbrains.com/is1/articles/04/10/lop/3.html`

composition of languages for the realization of systems.

1.4.1 Phase A: adoption of modeling and domain specific languages

Traditionally developers used one or more general purpose languages (GPLs) and a few supporting languages (HTML, SQL, etc.). Higher level languages as modeling and domain specific languages could be adopted to increase the level of abstraction. While those languages are reported to provide relevant benefits they are not yet mainstream. We therefore decided to study their adoption.

From the original broad research question *How can we adopt and combine specific languages for software development?* we focus on Phase A on the first part:

> How can we adopt specific languages for software development?

We decline this broad research questions on a set of more specific research questions:

- **RQ A.1** which are the benefits expected and attained from modeling and DSL adoption?
- **RQ A.2** which are problems limiting or preventing modeling and DSL adoption?
- **RQ A.3** which are the processes and techniques for modeling and DSL?
- **RQ A.4** which are the specificity of adoption of modeling and DSL in small companies?
- **RQ A.5** which are the specificity of adoption of modeling and DSL in large companies and software development ecosystems?

Through the first three RQs we plan to better *understand* these high level languages, while later, through the last two RQs, we plan to provide some insights to *guide* their adoption.

We decided to address the first three research questions through a survey among software development companies. That gave us the possibility to analyze a large and different number of different contexts, evaluating the effect of different aspects and benefits (RQ A.1) and problems (RQ A.2). We could explore also how the processes and tools adopted (RQ A.3) lead to different achievements and problems. The survey is presented in Chapter 2.

In our opinion the way techniques and processes can be adopted is influenced by the resources and the size of the company, therefore we performed two separate case-studies, one performed in a small companies (Chapter 3), while the second was performed in large company, at the center of a complex software development ecosystem (Chapter 4).

1.4.2 Phase B: combining multiple languages

Today most software development projects employs already multiple languages. We advocate the necessity of adopting even more specific languages. While the benefits of using the most suitable language for each task is addressed in the first part of the thesis, in the second we focus on a consequence of the choice of using an increasing number of languages to realize each single system: language interactions and the necessity to design appropriately language integration.

From the original broad research question *How can we adopt and combine specific languages for software development?* we focus on Phase B on the second part:

> How can we combine specific languages for software development?

We decline this broad research questions on a set of more specific research questions:

- **RQ B.1** how languages interact?
- **RQ B.2** which are the effects of language interactions?
- **RQ B.3** how can we identify language interactions?
- **RQ B.4** how can we offer tool support for language integration?

This area of research is very recent at the time this thesis is being written, so the phenomenon have to be investigated almost from scratch.

Similary to phase A, also for Phase B the initial RQs (RQ B.1 and RQ B.2) are devoted to *understanding* the phenomenon, while the later (RQ B.3 and RQ B.4) are devoted to provide *insights* and solutions.

To address RQ B.1 we manually analyzed artifacts written in different languages which evolved at the same time, compiling a classification of the different forms of interactions which link artifacts written in different languages (Chapter 5).

To start addressing RQ B.2 we performed a preliminary experiment to verify the effects of cross-language relations on defectivity (Chapter 6).

Performing this preliminar investigation on the phenomenon we decided to start focusing on the most common form of cross-language interactions: *shared-ID*. We studied the possibility of automatically spot cross-language relations (RQ B.3) by using heuristics and machine learning techniques (Chapter 7).

To address RQ B.4 we then realized a prototype of tool support for language integration using a language workbench (Chapter 8).

Finally all the results are summarized in the conclusions (Chapter 9).

Chapter 2

Relevance, benefits and problems of modeling and DSL adoption

We performed a survey in the Italian industry about adoption of model-driven development. Here we report the combined results published in [Torchiano et al., 2011b, Tomassetti et al., 2012, Torchiano et al., 2012, Torchiano et al., 2013]. It was a joint work realized with Marco Torchiano, Filippo Ricca, Alessandro Tiso, and Gianna Reggio.

Through the survey and the debriefing sessions we had with modeling experts we addressed some of the principal research questions of this thesis:

- **RQ A.1** which are the benefits expected and attained from modeling and DSL adoption?
- **RQ A.2** which are problems limiting or preventing modeling and DSL adoption?
- **RQ A.3** which are the processes and techniques for modeling and DSL?

We were able to answer additional research questions, not directly relevant for the main goal of thesis but still useful to acquire more knowledge about the phenomenon and drive successive research actions. In particular this survey permitted to obtain data bout the relevance of the phenomenon in Italy.

2.1 Introduction

Usually, model-based techniques use models to describe the architecture and design of a system and/or the behaviour of software artefacts generally through a raise in the level of abstraction [France and Rumpe, 2003]. Models can also be used for other purposes, e.g., to describe business workflows or development processes.

They can be used in different phases of the development process as communication artefacts, as points of references against which subsequent implementations are verified, or as the basis for further development. In the latter case, they may become the key elements in the process, from which other artefacts (most notably code) are generated.

The term "model" is very general, as we anticipated in the introduction of this thesis.

We are interested in no specific model driven technique, therefore, in the following, taking inspiration from [Völter, 2009], we will address all these related technologies collectively with the abbreviation MD*.

Although a lot of work has been done in the MD* context and, a wide number of studies have been reported in literature (e.g., [Hutchinson et al., 2011a, Hutchinson et al., 2011b, Mohagheghi et al., 2012]), and many commercial or free MD* tools are also available (e.g., UniMod [Gurov et al., 2007], WebRatio [Acerbis et al., 2007] and BridgePoint[1]), there is a lack of evidence about the relevance of MD* in industry, and we need indication whether (or not) MD* satisfies today's industry needs [Mohagheghi and Dehlen, 2010].

For those reasons, two Italian universities, Politecnico di Torino and Università di Genova, started a common project concerning software modelling and MD*. The first step of the project aimed at achieving an accurate picture of the state-of-the-practice of modelling and MD* in the Italian industry by means of a survey of the Italian ICT industry. We opted for a personal opinion survey[2] [Groves et al., 2009, Kitchenham and Pfleeger, 2008] performed through the Internet, because this is generally the most cost effective interview method [Walonick, 1997] even if it presents well-known limitations/problems [Singer et al., 2008].

The evidence we collected about modelling and MD*, by means of this survey, holds a value "per se" as new assets in the software engineering body of knowledge. In addition, it brings important implications in the practice of both software development and education/training. We think that, on the basis of the results of this broad survey, more specific studies could be conducted to confirm and clarify the most controversial or difficult understandable findings.

The chapter is structured as follows. Section 2.2 presents the relevant aspects of the conducted survey such as: goals, research questions, questionnaire design, sample identification, survey preparation/execution and analysis methodology. Findings are presented in three different sections: (Sect. 2.3) relevance of modelling and MD*, (Sect. 2.4) maturity of the approaches, (Sect. 2.5) benefits and problems. In

[1] http://www.mentor.com/products/sm/model_development/bridgepoint/

[2] The purpose of a personal opinion survey is to produce statistics, that is, quantitative or numerical descriptions of some aspects of the study population.

Section 2.6 we ask experts to comment some of the results which were more difficult to interpret. In Section 2.7 we discuss the obtained results. In Section 2.8 we examines the unavoidable threats to validity. In Section 2.9 we discuss related work and, finally, Section 2.10 summarize results from the survey and presents ideas for future work. Appendix 2.10 collect some detailed data which we could not discuss exhaustively in the rest of the chapter.

2.2 Study definition

The instrument we selected to take a snapshot of the state of the practice concerning industrial MD* adoption is that of a survey [Groves et al., 2009]. In the design phase of the survey we drew inspiration from previous surveys (i.e., [Jelitshka et al., 2007, Leotta et al., 2012, Torchiano et al., 2011a]) and we followed as much as possible the guidelines provided in [Kitchenham and Pfleeger, 2008].

The survey has been conducted through the following six steps [Kitchenham and Pfleeger, 2008]: (1) the objectives (or goals) selection, (2) goals' transformation into research questions (section 2.2.1), (3) questionnaire design, (4) sampling and evaluation of the questionnaire by means of pilot executions, (5) survey execution and, (6) analysis of results and packaging.

We conceived and designed the survey with the goals of understanding:

G1 the actual relevance of software modelling and MD* in the Italian industry,

G2 the way software modelling and MD* are applied (i.e., which processes, languages and tools are used), and

G3 the motivations either leading to the adoption (expected benefits) or preventing it (experienced or perceived problems).

In the context of this thesis G2 contributes to answer **RQ A.3** while G3 contribute to **RQ A.1** and **RQ A.2**. In the rest of the chapter research questions numbers will not refer to the general numbering of the thesis but to the numbering of the research questions specifically formulated for the survey.

2.2.1 Research Questions

Here we specify the research questions individuated for each goal.

Goal 1: relevance of software modelling and MD*

RQ1: What is the diffusion and relevance of modelling and MD* in the Italian industry?

Finding out the proportion of IT professionals actually adopting modelling and MD* should allow us to understand how important such development techniques are. Knowing whether they represent niche, commonly used, or mainstream methodologies has a dramatic impact on the conclusions we can draw.

RQ1.1: What is the adoption ratio of individual MD* techniques?

Since MD* is typically applied with diverse blends of basic techniques, we want to examine which techniques are used (e.g., code generation or model execution) and which are the most relevant.

RQ1.2: What is the diffusion by company size category?

Some authors (e.g., Selic [Selic, 2003]) believe MD* is mostly intended for large projects carried on by large companies; is it true? An empirical validation of such a claim requires breaking down the prevalence figures by company size.

RQ2: What is the experience in modelling and MD*?

From initial experimentations to proper mastery, different levels of maturity are possible. MD*, as all the software development approaches, present their own learning curve: at each level of maturity different outcomes can be derived. We are interested in understanding how much companies master MD* techniques and the effects of the maturity of the approaches on the outcomes obtained.

Goal 2: how software modelling and MD* are applied

RQ3: Which modelling languages and notations are used in the modelling phase and for MD*?

Modelling languages are various: they can have different nature (procedural, declarative, functional) and use different notations (textual, graphical, tabular), be general-purpose or domain-specific. We are interested in understanding which languages are used for which tasks and how the language chosen affect the outcome.

RQ4: What kind of processes and tools does Italian industry adopt to support modelling and MD*?

MD* included a large set of activities which can be combined to compose a concrete software development process. We think that the process and the supporting tools could have an important role in determining the outcome of MD* adoption. We therefore decided to investigate also this aspect.

RQ5: Which factors influence the maturity of modeling and MD*?

Which are the companies who employ more mature techniques? Are they large or small? Are they part of a particular domain? What kind of teams they employ?

Goal 3: motivations

RQ6: Which are the benefits expected from modeling and MD* adoption?

Expectations determine the choice of adopting technologies. We want to understand which are those expectations and if they are fullfilled or not.

RQ6.1: Which are the most expected benefits? We want to understand which are the anticipated benefits that also represent plausible motivations for adopting modeling and MD*.

RQ6.2: Which are the relations between expectations? We envision group of related benefits, i.e., benefits that are supposed to be achieved together.

RQ6.3: Which are the effects of experience on forecasting? Practicioners with more experience are better at forecasting the benefits of modeling?

RQ7: What are the benefits of using modelling and MD*?

From a technology transfer perspective it is important to motivate the adoption of a new technique by presenting the benefits that it could bring. Therefore, we are interested in understanding which specific technique (e.g., code generation) increases the likelihood of a given benefit.

RQ7.1: Do individual MD* techniques affect the achievement ratio of specific benefits?

RQ7.2: Which benefits are most common in each company size category?

RQ8: What issues hinder/prevent the adoption of modelling and MD*?

The reasons that hinder/prevent the adoption of modelling and MD* are as important as the potential benefits. In some cases these motivations may exclude such techniques from the whole company, in other cases the preclusion may be on a per-project basis.

2.2.2 Population and sampling strategy

The first step to conduct a survey consists in defining a target population. In our study the target population is formed by software development teams or business units. To get information about the target population we defined a framing population consisting of Italian software professionals – i.e., project managers, software architects, developers – whom we asked to answer our questions (about the target items).

To sample the population we applied both probabilistic (random sampling) and non-probabilistic (convenience sampling) methods [Kitchenham and Pfleeger, 2008]. More in detail, the sampling was performed in five ways: (i) randomly selecting contacts from the the Commerce Chamber database, (ii) selecting contacts from the industrial contact networks of the two research units involved (Torino and Genova), (iii) sending invitation messages to mailing lists concerning programming and software engineering, (iv) publishing an advertisement in an on-line magazine for developers (programmazione.it) and, and (v) placing and advertisement on the web portal of a large Italian developers' conference (CodeMotion 2011).

We decided to collect data through an on-line questionnaire created by means of the LimeSurvey survey tool[3]. Web-based questionnaires, compared to paper-based questionnaires or email-based questionnaires, allow an easier data entry from the respondent perspective, a simpler data collection from the researcher perspective and are less error prone [Punter et al., 2003]. In general, it has been observed that Web-based questionnaires guarantee high return rates [Jelitshka et al., 2007].

2.2.3 Survey Preparation and Execution

The procedure followed to prepare, administer, and collect the questionnaire data is made up of the following five main steps.

Preparation and design of the questionnaire. We first prepared a preliminary version of the questionnaire. Then, we conducted three different pilots with software professionals to identify any problems with the questionnaire itself [Kitchenham and Pfleeger, 2008], before putting on-line the final version. According to the feedback obtained, we made a few changes to the questionnaire to improve the validity of the instrument.

On-line deployment. Once finalized, the questionnaire was uploaded to the LimeSurvey server to enable the automatic collection of data.

Invitation to participate. Organizations were sampled as detailed above (Section 2.2). For the contacts that we selected directly, once the contact persons were identified, we invited them, via email, to register with the survey server and to complete

[3]http://www.limesurvey.org/

the on-line questionnaire. We also broadcast invitation on selected mailing lists and on-line magazines/conferences including in the message a link ("click here to take the survey") to a registration form where the participants could register themselves and fill in the questionnaire. The questionnaire was introduced by a brief description page summarizing goals and motivation of this study and it was accompanied by a cover letter briefly introducing our research project. In the cover letter we tried to summarize: the purpose of the study, the relevance to the participants and, why each individual participation was important [Kitchenham and Pfleeger, 2008]. Great care was taken to ensure that ethical requirements and privacy rules imposed by the Italian regulations were met[4]. For example, they require confidentiality but give the possibility of publishing the results in aggregated form. We decided to avoid any form of material incentives for participation. However, to motivate professionals, we promised to provide a report containing the analyses and the obtained results to all participants.

Monitoring. During the data collection phase, we monitored the progress of the questionnaire submission. This allowed us to send selective reminders to contacts who did not respond or did not completed the questionnaire yet. Some people reported some difficulties about the questions, either due of internal policies of the company or because involved in very different projects with different companies at the same time; they asked us some clarifications about the questions.

Data analysis and Packaging. After data collection was concluded, we performed analyses as described in section 2.2.5 and we packaged instruments, data, and results in a replication package.

2.2.4 Questionnaire Design

The questionnaire was developed to directly address the goals of the study. To harvest as many responses as possible, we designed the questionnaire to limit as much as possible the time necessary to complete it, having in mind that long questionnaires get less response than short ones [Walonick, 1997][5].

The questionnaire contains a total of thirty-one items, both open and multiple-choice expressed in Italian language. However, the actual number of items administered to any individual respondent depends on her adoption of MD* and modelling (e.g., respondents not adopting modelling in their software process were required to answer eight questions only).

The structure and possible paths through the questionnaire are described in

[4]privacy Italian law: "D.lgs. n.196/2003".

[5]It turned out the actual time for completing the questionnaire was on average less than six minutes.

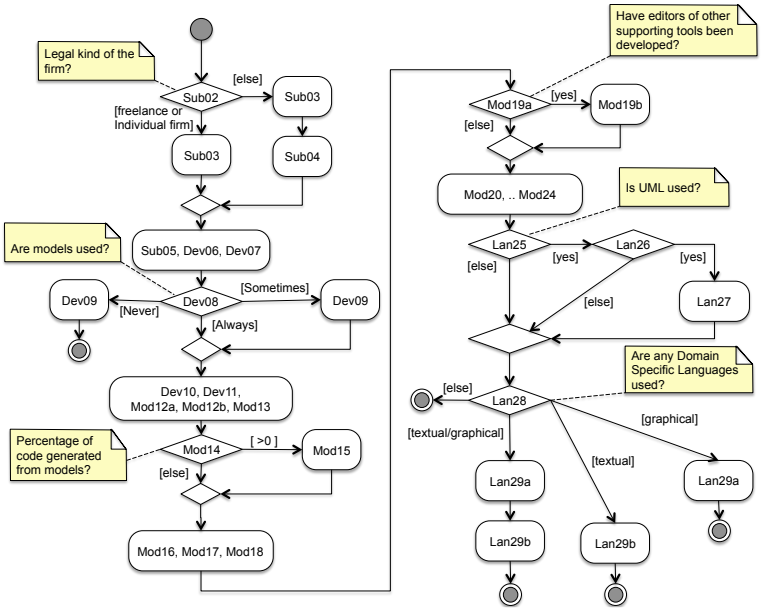

Figure 2.1. Questionnaire structure.

Figure 2.1. The questionnaire consists of four sections; each session is identified by a three characters identifier (Sub, Dev, Mod, and Lan) and is described below. The individual items are named using the section identifier followed by a two digits progressive number (e.g., Sub04 is the fourth item in the questionnaire and appears in the Sub section).

Sub (subject's demographics): this is the first section and is administered to all respondents; its goal is to characterize the respondents and their organization. In particular, it collects: business domain, organization size, respondent's group/business unit size and experience of unit members. For example, freelance or individual companies are asked only three distinct questions of this kind (Sub02, Sub03 and Sub05), as we may observe from Figure 2.1.

Dev (development process): the second section collects information concerning the kind of projects conducted, their average duration, whether the respondent uses models, and the expected and achieved benefits. The respondents that do not use modelling systematically (i.e., answering *never* or *sometimes* at question Dev08) are asked about the problems preventing or limiting to use models (see question Dev09 in Table 3.1 and Figure 2.1). Respondents never using modelling terminate the questionnaire with this section.

Mod (modelling details): the items in this section are administered to respondents

that use modelling at least sometimes. The section collects information concerning the adoption of processes, techniques and tools.

Lan (languages and notations): contains items measuring use of UML, UML profiles and domain specific languages (DSLs).

Table 2.1 reports the items in the questionnaire. For each question the table reports the identifier, the question (translated into English from Italian) and the type of measure.The complete questionnaire (in Italian) is available for downloading on the web[6].

ID	Question	Type
Sub02	What legal entity/kind does your company fit? *Valid answers: Freelance/individual firm; Firm/company; Public institution; Other*	Nominal
Sub03	What is the main business activity of your company? *Valid answers: [Manufacturing; IT; Public Administration; Service Provider; Transport; Telecom; Other]*	Nominal
Sub04	How many persons does your company count, including part-time, full-time staff and consultants?	Ordinal
Sub05	Provide the experience (in years) of the business unit's members	Interval
Dev06	What is the typical size of the development team?	Preset intervals
Dev07	What is the typical effort in person-months per project?	Interval
Dev08	Are models used for software development in your organization? (for model we mean both diagrams, e.g., UML, and text according to any DSL) *Valid answers: Always; Sometimes; Never*	Nominal
Dev09	What are the problems preventing modelling and MD*?	Nominal
DEV10	For which goals was decided to use modelling?	Nominal
Dev11	What are the benefits expected and verified from using modelling and MD*?	Nominal
Mod12a	How many elements diagrams contain?	Interval
Mod12b	How many lines DSL utterances contain?	Preset intervals
Mod13	Who write the models? *Valid answers, not exclusive: Developers; Project managers; Business experts*	Nominal ?
Mod14	What is the percentage of code generated from models?	Interval
Mod14*	Is code generated? *Derived measure:* $Mod14* = Mod14 > 0$	Yes/No
Mod15	For which layes is code generated?	Nominal
Mod16	Are models executed (interpreted) at run-time?	Yes/No
Mod17	How many metamodels are typically used per project?	Interval
Mod18	Are transformation languages (e.g., ATL) used?	Yes/No
MD_USAGE	Derived from previous: $MD\_USAGE \leftarrow (MOD14 > 0) \vert MOD16 \vert MOD18$	Yes/No
Mod19a	Have modelling support tools and editors been developed?	Yes/No
MD_USAGE	Is any MD* technique used? *Derived measure:* $MD\_USAGE = Mod14* \vee Mod16 \vee Mod18$	Yes/No
Mod19b	Using which technologies?	Nominal
Mod20	Are versioning systems used?	Yes/No
Mod21	How many years ago was modelling adopted?	Ordinal
Mod22	In how many projects has modelling been used in the last 2 years?	Percentage
Mod23	In how many projects has MD* been used in the last 2 years?	Percentage
Mod24	How many years ago was MD* adopted?	Ordinal
Lan25	Is UML used?	Yes/No
Lan26	Are UML profiles used?	Yes/No
Lan27	How many stereotypes are used in UML Profiles?	Interval
Lan28	Are Domain Specific Languages used? *Valid answers: No; Yes, textual; Yes, graphical; Yes, both graphical and textual*	Nominal
Lan29a	How many node types are used in graphical DSLs?	Preset interval
Lan29b	How many constructs are used in textual DSLs?	Preset interval

Table 2.1. Questionnaire items considered (translated from Italian to English). Questions are condensed in respect to the administered survey.

A few items in the questionnaire are particularly important for the purpose of this study and deserve more attention.

In section Dev, the most important item is Dev08 that corresponds to the following question: "*Are models used for software development in your organization?" By*

[6] *http://softeng.polito.it/tomassetti/MDQuestionnaire.pdf*

model we mean both diagrams and text artefacts created using either general purpose modelling languages (e.g., UML) or Domain Specific Languages (DSLs)[7]. Dev08 is a closed question that allows three valid answers: *always*, *sometimes* and *never*. Depending on the Dev08 answer, three distinct paths were followed (see questionnaire structure in Figure 2.1):

- respondents that *always* use modelling were asked about the benefits (Dev11),
- those using modelling only *sometimes* were asked about the benefits (Dev11) and also about the problems preventing the use of modelling (Dev09), and
- respondents *never* using modelling were asked only about the potential problems (Dev09).

Given their importance, we evaluated accurately the possible answers presented for questions Dev09 and Dev11 in the design phase of the questionnaire. Moreover, we fine-tuned them based on the outcomes of the pilots. The values we chose are reported in Figure 2.2.

Since we were not sure about the range of possible problems preventing the usage of models and MD*, we designed item Dev09 as a multiple answers question with a set of predefined options plus an additional open option.

Conversely, after the feedback from the pilots, we were confident we identified all the significant benefits, therefore item Dev11 was designed as a closed answer with multiple choices. The question Dev11 asks which benefits among the ones presented were expected by the respondents and which ones were actually verified. For each benefit the respondent had the possibility to mark separately if the benefit was expected or verified. In this way four combinations are possible for each single benefit: it could have been expected and verified (positive confirm), expected and not verified (negative surprise), not expected and verified (positive surprise) or not expected and not verified (negative confirm).

In the questionnaire there are three items concerning MD*-specific practices: code generation (Mod14), model interpretation (Mod16), and model transformations (Mod18). We plan to handle the three practice categories in an homogeneous way; since Mod14 measures the percentage of code generated from models, while the other two (Mod16 and Mod18) just measure the adoption as a boolean value, we introduced a derived variable (Mod14*) with boolean type whose value is true when some code is generated and false otherwise, i.e., $Mod14* = Mod14 > 0$.

We assume that using at least one of the three techniques means adopting MD*, therefore, we defined a derived item (MOD_USAGE see Table 3.1) that is defined

[7] *This was clarified in the questionnaire given to the participants.*

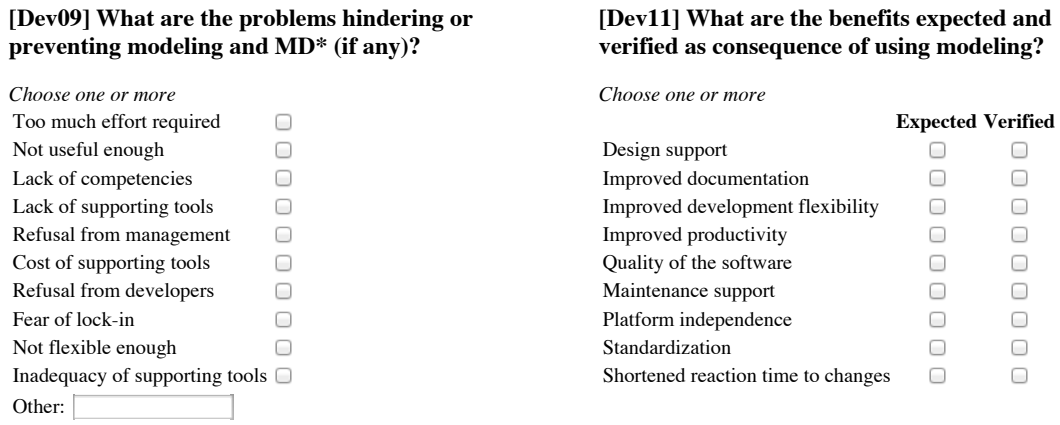

Figure 2.2. Options presented in questions Dev09 and Dev11.

on the basis of the three technique oriented items, i.e., $MD\_USAGE = Mod14* \vee Mod16 \vee Mod18$.

2.2.5 Analysis methodology

We address the three research questions delineated in Section 2.2.1 by means of descriptive statistics and where applicable by statistical hypothesis testing, and show our findings by means of graphs.

Statistical correlation between a factor (e.g., using UML profiles) and the benefit achievement ratio (the proportion of respondents who achieved each specific benefit) are verified by means of standard tests. In particular, we opted for non-parametric tests because of the nature of the variables, which are measured on nominal and ordinal scales. Specifically, according to the recommendations given in [Agresti, 2007, Motulsky, 2010], we used:

- the Fisher exact test for the correlation among two dichotomous variables,
- the χ^2 test for the correlation among categorical variable variables,
- the Mann-Whitney tests for testing the difference between two groups (we can interpret a significant MW test as showing a difference in medians), and
- the Kruskal-Wallis test for three or more groups.

The decision whether rejecting or not the null hypotheses verified by statistical tests is taken considering a level of significance of 95%. This means that when we

draw our conclusions, we accept a probability ($\alpha = 5\%$) of incurring in a type I error (i.e., rejecting a null hypothesis when it is true).

We present now the method used to specifically address each research question.

RQ1: diffusion and relevance

We answer RQ1 by looking at one dependent variable: the answers to the item about model usage (Dev08). In particular, we focus on the frequency of the three valid answers (*Never*, *Sometimes*, and *Always*). The analysis of both this RQ and the following ones, will focus on the "modellers", with this term hereinafter we refer to the respondents who use modelling at least sometimes: that is those who answered *Sometimes* or *Always* to item Dev08.

In agreement with the definition of relevance provided in [Hjørland and Sejer Christensen, 2002], we can state that the study of a specific software development technology is relevant to software engineering if it increases the likelihood of improving software development practices. Such a perspective involves both technical aspects, which are out of the scope of our investigation, and process aspects which are in part addressed in our investigation. With a little bit of simplification we assume here that the main process factor for evaluating the diffusion of a technology is the proportion of developers that use it. In the context of this study, we assume a proportion larger than 50% implies a high relevance, larger than 25% a normal relevance, larger than 10% a limited relevance, and below 10% irrelevance. We defined the above thresholds in an arbitrary way using just common sense.

We compute the relevance level (high, normal, limited, irrelevant) for modelling by comparing the above thresholds to the confidence intervals for the proportions. In particular, we compute the 95% confidence interval (CI) by means of the proportions test [Agresti, 2007]. Then, to be conservative as much as possible, we assign the level corresponding the highest threshold that is smaller than the lower limit of the confidence interval. For instance if the 95% CI of the diffusion of a technology is [35% , 65%] we assign the "normal relevance" category since the corresponding threshold (25%) is the highest one smaller than the lower limit (35%).

Then, we analyse the relevance and diffusion indicators with respect to the company size (Sub04). In particular, we check for the existence of a correlation between the adoption ratio of modelling and the company size categories. To this end, we build a 2×5 contingency table of modelling adoption (sometimes or always) vsṅo modelling on one side and company size categories on the other side (we considered five company size categories, see Table 2.2). We apply the χ^2 test to reject the null hypothesis that no correlation exists. In addition we evaluate the relevance for each company size category

Moreover, we focus on the adoption of the MD* specific practices: code generation (Mod14*), model interpretation (Mod16), and model transformations (Mod18).

In particular, we assume that respondents using at least one of the above cited techniques is adopting MD*.

For both the general MD* adoption and each specific technique we perform the same analyses as above: first we classify the relevance and then we analyze it in relation to company size.

RQ2: experience in modelling and MD*

Four items in the questionnaire are devoted to the experience dimension (see Table I). They capture information regarding modelling and MD* employees experience measured in years (staff experience) and percentage of projects where modelling and MD* are adopted respectively (organization experience).

We therefore simply considered years of experience in basic modeling and MD*

RQ3: languages and notations used

We considrered how many professionals use UML, with or without profiles and how many use DSLs, differentiation between textual and graphical.

RQ4: processes and tools used

We examined which role perform the modeling, which MD* techniques are used among: model-interpretation, code-generation, model-to-model transformation. We also verified who developed their own tools and who use versioning for models.

RQ5: factors influencing maturity

We used as categorical criteria (i) company size and (ii) years of experience in modeling. We observed if these criteria affected positively some selected indicators of maturity: Mod14* (Code generation), Mod16 (Model execution), Mod18 (Transformations), Mod19a (Specialized editors) and Mod20 (Versioning).

We represented the effects of these criterias on these factors by means of polar charts.

RQ6: benefits expected

RQ6.1: to answer this RQ we simply ranked the benefits by the number of respondents expecting that benefit in descending order. In addition, using the proportion test, we compute the estimate proportion of respondents who expect the benefit and the corresponding 95% confidence interval. The interval is useful to understand the precision of the result.

RQ6.2: we looked at the relations between all possible pairs of benefits. We calculated the Kendall rank correlation coefficient between the expectations of each pair of benefits, obtaining a symmetrical measure of the strength of association between the expectations of the two benefits. Positive values represent a positive association while negative values represent a negative association. The absolute value of the correlation represent the strength of the association and it can vary from zero to one.

RQ7: benefits achieved

Question RQ7 is addressed by analysing the answers to a composite item (Dev11) which listed several potential benefits and asked which were expected and which were achieved (see Figure 2.2). For this question we focus on the actually achieved advantages. In particular we adopt as metric the benefit achievement ratio: the proportion of respondents who achieved each specific benefit.

As a preliminary step, we investigate the cross correlation among the different benefits, since the adoption is a dichotomous variable, we selected the Pearson's ϕ as a strength of correlation measure. The statistical significance of the correlation is verified by means of the Fisher exact test.

We first report the benefit achievement ratio among all modellers, with the intention of classifying the benefits in terms of their likelihood. In particular we assume that above a 50% frequency a benefit can be considered as *Very Likely*, above 25% as *Likely*, above 10% as simply *Probable*, and below that threshold as *Unlikely*. As for the relevance categories of RQ1, we compute the 95% confidence interval of the proportion using a proportion test and compare the lower limit with the above thresholds.

Then, we report the proportion of respondents who achieved each specific benefit making a distinction between adopters of simple modelling and adopters of MD*. We investigate whether a significant difference in benefit achievement ratio between the two groups exists and for any such case we will perform a further classification of the relative likelihood. For the purpose of identifying the difference we observe the odds ratio[8] of benefits achievement for the two groups and test the significance by means of the Fisher exact test.

[8]The odds ratio is a measure of effect size that can be used for dichotomous categorical data. An odds indicates how likely it is that an event will occur as opposed to it not occurring. Odds ratio is defined as the ratio of the odds of an event occurring in one group to the odds of it occurring in another group. An odds ratio of 1 indicates that the condition or event under study is equally likely in both groups. An odds ratio greater than 1 indicates that the condition or event is more likely in the first group. Finally, an odds ratio less than 1 indicates that the condition or event is less likely in the first group.

After that, we focus on the three key MD* techniques (i.e., code generation, model transformation and model interpretation). In particular, we check whether the adoption of a single technique induces a significant difference in terms of benefit achievement ratio. As for the previous step, we focus on odds ratios and we use the Fisher exact test to identify significant differences.

Then, we check for the existence of a correlation between the achievement ratio of each individual benefit and the company size categories. Similarly to the procedure adopted for RQ1, for each benefit, we build a 2×5 contingency table of achievement vs. not achievement of a benefit on one side and company size categories on the other side. We apply the χ^2 test to reject the null hypothesis that no correlation exists.

Eventually, as a last step concerning RQ7.1, we observe the relationship between benefit achievement and other factors, namely toolsmithing, adoption of UML, UML profiles, and DSLs. In particular we observe how the odds of achieving a benefit change when each individual technique is employed. The Fisher exact test is used to test the null hypothesis that no significance difference exists.

RQ8: problems

To answer RQ8, we consider the composite item Dev09 which reports for each participant a list of problems (s)he found prevented her/him from adopt modelling (see Figure 2.2). The respondents to this item include those who never use modelling and those who use it sometimes: for the former the problems prevent altogether modelling, while for the latter the problems just curb it.

We adopt a similar approach as for RQ7, therefore we first verify cross correlation among problems, then we report the problem occurrence ratio among modellers, with the intention of classifying them in terms of relevance. For this purpose we adopt the same criteria used in RQ1.

In analogy with the procedure adopted for RQ7, we eventually divide the respondents into two groups – those who never adopted modelling and those who used it sometimes – and we analyse the problems considering their relationship with company size.

2.3 Findings about relevance of modelling and MD*

In this section we first describe the sample (Sect. 2.3.1), then we present findings relative to RQ1: relevance and diffusion (Sect. 2.3.2), and RQ2: experience level (Sect. 2.3.3).

2.3.1 The sample

The survey was put on-line from the 1st of February 2011 until the 15th of April 2011 (two and a half months).

In total, we collected 155 complete responses to our survey, thus the context of our survey consists of a sample of 155 Italian software professionals. Due to the sampling methods used, it is not possible to estimate how many people have been reached by our invitation messages and/or advertisements; as a consequence it is not possible to compute the response rate. This limitation appears to be a common problem for large scale online surveys (see, e.g., [Lethbridge, 1998]).

The most of the companies where the respondents work are in the IT domain (104), then come services (15) and telecommunications (11). The distribution of the companies size where the respondents work is presented in Figure 2.3.

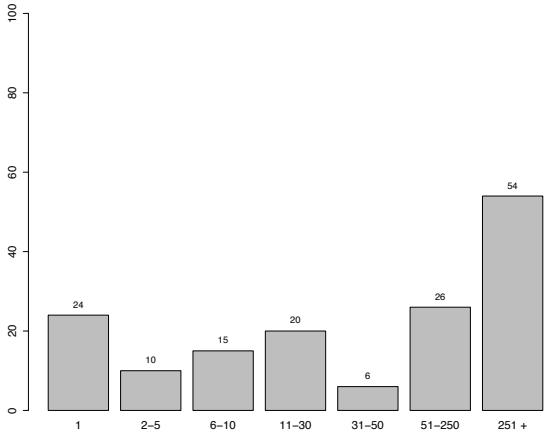

Figure 2.3. Size of respondents' companies

Among the respondents, on the basis of item *Dev08* we were able to identify 105 respondents using modeling and/or MD* techniques.

Interval	Group	Frequency
1	Individual	24
2-10	Micro	25
11-50	Small	26
51-250	Medium	26
251+	Large	54

Table 2.2. Frequency of respondents for different company sizes.

As far as the type of company is concerned, most respondents in our sample – 122 out of 155 (78.71%) – work in commercial companies; there are also 24 independent professionals (15.48%), six from public organizations (3.87%) and three from other

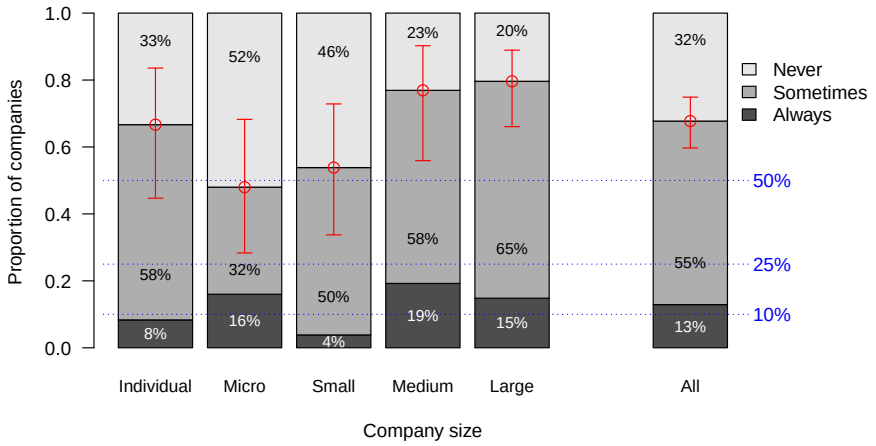

Figure 2.4. Proportion of modeling usage per company size.

organizations. Concerning the domain, the most represented sector is obviously *IT* (67%) followed by *Services* (9%), *Telecommunications* (6%) and *Manufacturing* (4%); the remaining sectors all together account for circa 14% of the sample.

The respondents to our survey belong to companies of different size; the detailed distribution for each category class is reported in Table 2.2. We adopted the headcount classes defined in the European Union recommendation 2003/361/EC[9]

It is important to emphasize that in our sample the correspondence between respondents and companies is not strictly one-to-one. In some cases we had more than one response from employees of the same company: in such cases we verified that they worked in distinct business units. This is obvious when consider that the target population consists of development teams and companies, especially the large ones, typically host several business units and work groups, each possibly working in different settings.

2.3.2 RQ1: relevance and diffusion

We first consider relevance of modelling in general, then we summarize languages and notations used by our sample. Finally, we examine the diffusion of each specific MD* techniques.

[9] `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?\uri=OJ:L:2003:124:0036:0041:EN:PDF`

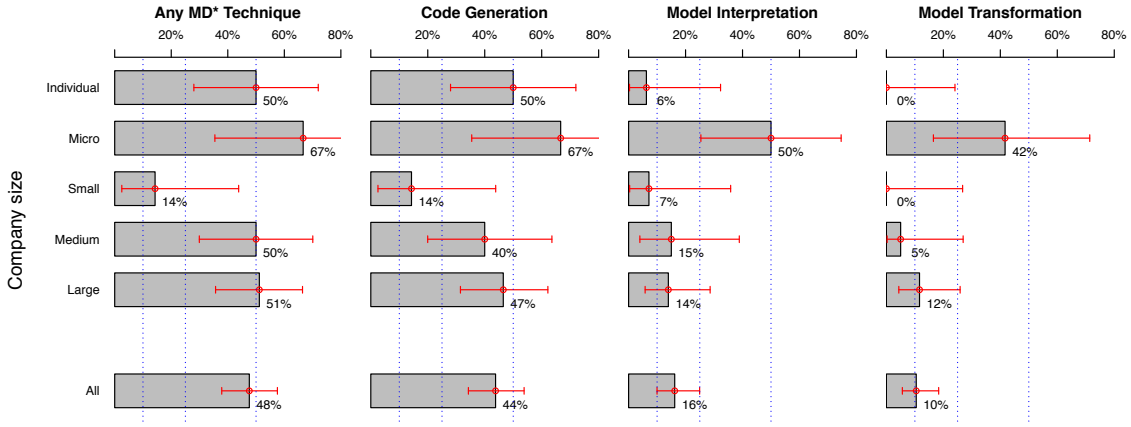

Figure 2.5. Diffusion of MD* techniques among modellers per company size.

Relevance of modelling in general

Among the 155 complete questionnaires, we recorded 20 respondents (13%) always using modeling, 85 (55%) using it sometimes, and 50 (32%) who never use it. The first two groups represent what we called "modellers". These figures are reported in the rightmost bar of Figure 2.4. The circle represents the estimate proportion of modellers in the sample (68%) and the whiskers represents the 95% confidence interval (CI) of the same proportion.

We can classify the relevance of modelling by comparing such CI with the thresholds defined in Section 2.2.5 (50%, 25%, and 10%), which are represented by the horizontal dashed lines. The 95% CI of the proportion of modellers is [60%, 75%], the lower bound is larger than the 50% threshold, therefore modeling can be classified as a *highly relevant* technology. We can get to the same classification by graphically comparing the lower end of the CI with the reference dashed lines: the whiskers lie completely above the 50% reference.

The proportion of developers adopting modelling varies significantly (χ^2 test p = 0.02) with the size of company as we can appreciate also in Figure 2.4. According to the 95% CI we can still classify modeling as highly relevant for medium and large companies, while it can be considered as simply relevant for the other companies.

We observe that, with the exception of "individual" companies, the use of modelling (i.e., always + sometimes) is positively correlated with the size of companies (i.e., it is more frequent in large companies). Individual companies represent an exception to that trend since they are closer to large companies' levels. Concerning the systematic use of modelling (i.e., always), we observe a similar diffusion at micro, medium, and large companies; while apparently there are few small-sized companies

and individuals that systematically adopt modelling practices.

Languages and notations

Among the 105 modellers, 80 of them (76%) adopt UML as modelling language (Lan25). Among them (Lan26), 11% use also UML profiles, 51% do no use them, and the remaining 38% state to not know if them are used in their organization.

In our sample, only 21% of modellers appear interested in Domain Specific Languages (Lan28). Among them 50% use a purely textual notation, 23% a purely graphical one, and 27% a mix of textual and graphical notations.

Relevance of MD* specific techniques

Among the 105 respondents that use modelling, 50 of them (48%) adopt at least one of the three key MD* techniques. The 95% CI of the proportion of developers among using MD* is [25% , 40%] of all developers, therefore we can classify MD* as a *relevant* development technology.

The relative frequency of adoption of the MD* specific practices among modellers is depicted in Figure 2.5. Overall, we observe that, in our sample, code generation is in use by 44% of the 105 modellers, model interpretation by 16%, and model transformation by 10% (not in exclusive way). The 95% CI of the frequency of the use of the individual techniques by all developers are [34% , 54%] for code generation, [10% , 25%] for model interpretation, and [6% , 18%] for model transformation. We can compare them to the relevance thresholds defined in section 2.2.5. Therefore code generation can be classified as a *relevant* technology, model interpretation as a technology with *limited relevance*, while the diffusion of model transformation is not enough to consider it relevant for the practitioners.

If we narrow down our scope to MD* adopters, only, 46 out of 50 MD* adopters (92%) use code generation, 34% use model interpretation, and 20% use model transformation.

2.3.3 RQ2: experience level

We differentiate between staff general experience and organization experience with modeling and MD*.

Staff experience: 40% of users using modelling in our sample have an experience in the range (2,5] years and 30% in the range (5,10] (Mod21). The experience in MD* is lower: 65% among modellers have no experience at all and 20% have 2 years or less of experience (Mod24). Figure 2.6 (middle) shows the complete distributions for experience in modelling and MD*. The experience in modelling appears to be distributed according to a normal distribution centred around the interval of

(2,5] years of experience. Instead, the experience in MD* has a distribution strongly skewed towards the zero.

Organization experience: The teams of our sample use modelling in 44% of the projects on average (Mod22)[10]. Adopters of modelling use MD* in 39% of the projects on average (Mod23). In both cases the time-frame considered is the last two years. Both modelling and MD* are used in more projects as the experience of the respondents in the field grows (see Figure 2.6, up for modelling and down for MD*). For example, see Figure 2.6 (up), modellers with an experience in the range (0,2] years adopt modelling only in the 20% of the projects (median) while modellers with more than 10 years of experience adopt it in the 80% of the projects (median).

The correlations between years of experience and proportion of projects adopting modelling and MD* respectively are statistically significant. In both cases the Kruskal-Wallis test returned a p-value < 0.001.

[10]The values were calculated considering only users with more than zero years of experience respectively in modeling and MD*

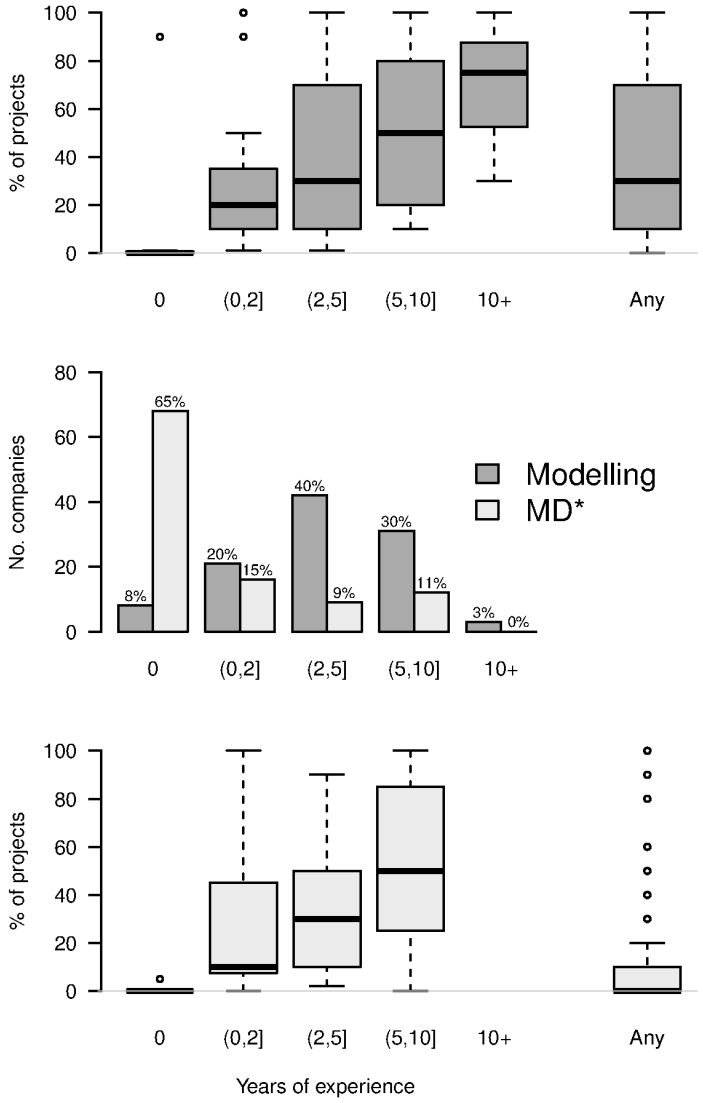

Figure 2.6. Staff (middle) and Organization experience (up and down) in modelling and MD*

		Achievement ratio			MD* vs. Basic modeling		Code generation		Model interpretation		Model transformation	
Benefit	Freq.	Estimate	95% CI	Likelihood	OR	p	OR	p	OR	p	OR	p
Design	71	68%	58% .. 76%	Very likely	1.23	0.68	1.0	1.00	2.5	0.26	1.3	1.00
Documentation	65	62%	52% .. 71%	Very likely	0.62	0.31	0.6	0.22	0.9	0.79	0.5	0.33
Maintenance	43	41%	32% .. 51%	Likely	1.49	0.33	1.2	0.69	1.8	0.29	2.8	0.12
Quality	42	40%	31% .. 50%	Likely	2.22	0.07	1.8	0.16	1.9	0.28	2.9	0.11
Standardization	39	37%	28% .. 47%	Likely	2.44	**0.04**	2.2	0.07	1.6	0.42	1.5	0.53
Flexibility	24	23%	15% .. 32%	Possible	2.17	0.11	1.4	0.49	**3.9**	**0.02**	2.1	0.27
Productivity	23	22%	15% .. 31%	Possible	5.53	< **0.01**	**3.9**	**0.01**	**4.2**	**0.01**	**8.3**	<**0.01**
Reactivity to changes	20	19%	12% .. 28%	Possible	1.84	0.32	1.1	1.00	**4.0**	**0.02**	2.7	0.21
Platform independence	15	14%	8% .. 23%	Unlikely	5.39	**0.01**	3.0	0.09	**4.7**	**0.02**	**4.2**	**0.05**

Table 2.3. Benefits achieved by modelling users (OR=Odds Ratio, p=Fisher test p-value).

	Toolsmithing		UML		UML Profile		DSL	
Benefit	OR	p.value	OR	p.value	OR	p.value	OR	p.value
Design	1.2	1.00	2.4	0.09	4.1	0.25	1.3	0.62
Documentation	0.5	0.18	2.6	0.06	1.1	1.00	0.9	0.81
Maintenance	2.4	0.11	2.1	0.16	1.9	0.46	1.0	1.00
Quality	1.9	0.28	1.2	0.82	1.1	1.00	1.3	0.63
Standardization	**3.9**	**0.01**	1.7	0.35	4.7	0.05	0.7	0.63
Flexibility	**3.9**	**0.02**	0.9	1.00	1.0	1.00	**3.1**	**0.04**
Productivity	**4.2**	**0.01**	1.2	1.00	1.2	1.00	**3.4**	**0.02**
Reactivity to changes	**5.5**	**< 0.01**	0.9	1.00	1.0	1.00	2.5	0.12
Platform independence	**9.9**	**< 0.01**	0.8	0.75	3.5	0.22	**4.3**	**0.01**

Table 2.4. Effects of additional factors on benefit achievement rate.

2.4 Findings about how software modelling and MD* are applied

In this section we present the findings related to RQ3: languages and notations (Sect. 2.4.1), RQ4: processes and tools (Sect. 2.4.2), and RQ5: factors affecting maturity (Sect. 2.4.3).

2.4.1 RQ3: languages and notations

We found that 76% of professionals (80 out of 105) creating models use UML; among them 11% use also UML profiles, 51% do no use them, and the remaining 38% state to not know if them are used in their organization.

In our sample, only 21% of professionals using models appear interested in Domain Specific Languages (DSLs). Among them 50% use a purely textual notation, 23% a purely graphical one, and 27% a mix of textual and graphical notations. Results are shown in Fig. 2.7.

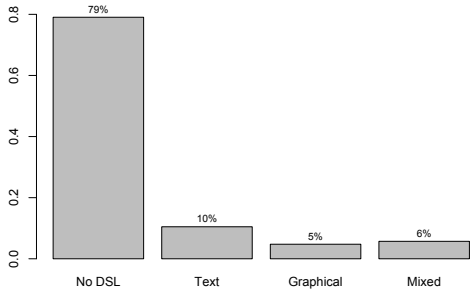

Figure 2.7. Usage and type of DSLs.

2.4.2 RQ4: processes and tools

As far as processes are concerned, we investigated which role typically performs the modelling. Usually modelling is performed by multiple roles at the same time. For this reason, to the corresponding question in the questionnaire several answers were permitted (e.g., developer and project manager). Figure 2.8 shows that architects or project managers perform modelling in 76% of the cases, developers write models in 72% of the cases, while domain experts are involved in just 11% of the cases.

As far as model manipulations, it appears the only 10% of modelers perform automatic transformations between models. While 16% of the modelers developed

editors or other support tools for models. Since models can evolve, 53% of the modelers adopt versioning of models.

Typically modeling is performed by multiple roles at the same time; table 2.5 breaks down the combination of different roles. We observe that when experts are involved (top two data rows) they consistently cooperate with high profile roles (Architects or PMs). While when experts are not involved, half of the times developers and architects operate together and half of the times they work alone.

		Developer	
	Architect	Yes	No
Expert	Yes	6	6
	No	0	0
No Expert	Yes	45	23
	No	25	-

Table 2.5. Roles performing modeling

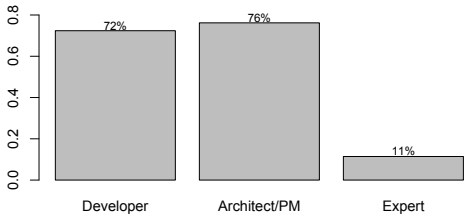

Figure 2.8. Which role writes the models.

When we focus on the adoption of (any) MD* specific techniques as a function of the company size we observe a sort of bimodal distribution. A very large proportion (67%) of the micro companies adopt MD* techniques, medium and large companies have an adoption ratio of circa 50%. This trend is completed by a sudden drop in adoption for small companies where just 14% adopt such techniques (see Figure 2.5 leftmost plot).

Considering each technique alone, we observe a similar shape for the distribution but with some notable differences. The adoption of code generation is very similar to the adoption of MD* techniques: this is obvious since it is by far the most widespread among the three techniques. As far as model interpretation and model transformation are concerned, micro companies are the only significant adopter, large companies adopt them very marginally and still small sized companies exhibit little or no interest in them.

The above picture can be drawn considering the adoption of the techniques as

separate; in practice specific techniques are adopted both individually ($^2/_3$ of the cases) and in combination with each other ($^1/_3$ of the cases).

Code Generation	○	●	○	○	●	●	○	●
Model Interpretation	○	○	●	○	●	○	●	●
Model Transformation	○	○	○	●	○	●	●	●
Freq	55 52%	30 28%	3 3%	0 0%	6 6%	3 3%	1 1%	7 7%

Table 2.6. Combined diffusion of MD* techniques (●: technique used, ○: technique not used).

Table 2.6 reports the relative frequency of the different combination of techniques (indicated in the table with black circles) that were found in use among the respondents. We observe that the most common toolbox consists of code generation alone (28% of modelers), the next most frequent sets are the combination all the three techniques (7%) and the use of code generation together with code model interpretation (6%). The other options are adopted by a few respondents. Notably, model transformation techniques are never used alone but only together with other techniques (that is pretty obvious).

2.4.3 RQ5: factors affecting maturity

Among the indicators we analysed above (see Table 3.1), we identified a subset of them to be further investigated, having in mind the goal of observing how different types of companies score in terms of maturity. As categorization criteria, we used: (i) company size and (ii) years of experience in modelling.

We decided to consider only boolean indicators (Yes/No answers) since they enable an immediate quantification of the maturity level of a group of companies: the percentage of companies having a positive indicator provides a measure of maturity in that group. Conversely, we discarded, for this analysis, the indicators measured with interval, nominal and ordinal metrics. For the interval metrics it is difficult to define one or more thresholds to assess the maturity level, e.g., is 42% of average code generated low or high? The same reasons hold, all the more so, for ordinal and nominal metrics.

Finally, the selected indicators are: Mod14* (CodeGeneration), Mod16 (ModelExecution), Mod18 (Transformations), Mod19a (Specialized editors), and Mod20 (Versioning).

As a tool to analyse the maturity from those five perspectives at once, we used polar charts. We recall that each line in a polar-chart represents a quantitative

synthesis of maturity. It was calculated as the mean of the values for that indicator between the respondents of a given group.

Figure 2.9 (left) shows the maturity along the five dimensions for companies grouped into different ranges of size, each size category corresponding to a different line style. By looking at the enclosed areas, micro companies (10 employees or less) appear to use a more mature approach in MD* than larger companies (indeed the area is bigger). The percentage of companies adopting code generation varies significantly as the size of company: micro companies and large companies, the latter to a lesser extent, adopt code generation more often than small and medium companies. The same applies to model transformations. In terms of model execution and specialized editors, micro companies perform better than the larger-sized companies. Finally, in terms of model versioning large companies adopt it slightly more than micro companies, while small and medium sized companies embrace it half as frequently.

Figure 2.9 (right) shows the five indicators based on the experience in modelling. The picture here, is less clear cut. Companies in the (5-10] range adopt slightly more often model transformations and execution and more often code generation than companies in the 10+ range; while the latter developed more often specialized editors. Companies with shorter experience adopt such techniques less often. Finally, we observe a natural evolution, as experience grows, from a reduced adoption of versioning up towards more diffuse adoption.

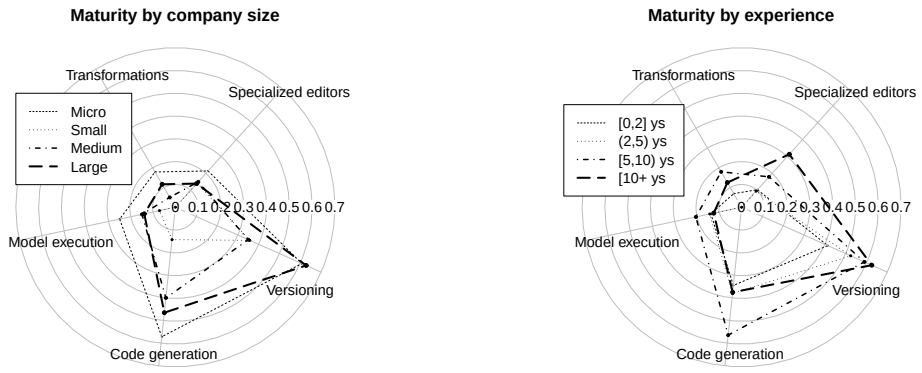

Figure 2.9. Maturity with respect to company size (left) and experience in modelling (right)

2.5 Findings about benefits and problems

In this section we present findings relatives to RQ6 (Sect. 2.5.1), RQ7 (Sect. 2.5.2), and RQ8 (Sect. 2.5.3).

2.5.1 RQ6: benefits expectations

In this section we present results about benefits expectations. We first report how frequently the different benefits were expected, then how these relations where expected, i.e. which group of benefits were commonly expected to be achieved together and finally the effects of experience on forecasting.

RQ6.1: Which are the benefits expected from modeling adoption?

In Table 2.7 we report for each benefit the frequency of expectation (column *Freq.*) and the corresponding percentage of respondents (column *Estimate*).

Improved documentation is the most expected benefit, with almost 4 out of 5

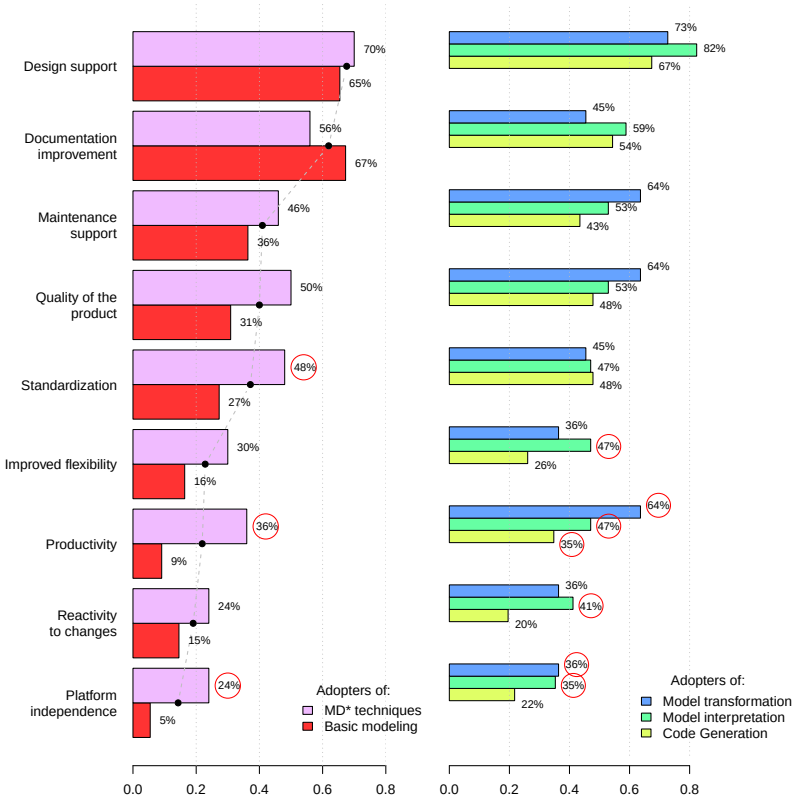

Figure 2.10. Benefits achieved. By "Basic modelling" we mean use of models not resorting on any MD* technique. Circles indicated statistically significant difference.

respondents anticipating it. Also *Design support*, *Quality of the software*, *Maintenance support*, and *Standardization* are frequently expected. For all of the top 5 benefits we are 95% sure that more than 50% of modeling adopters expect them: in fact the confidence interval (C.I.) lower bounds are larger than 50%. The remaining benefits, *Improved development flexibility*, *Improved productivity*, *Shortened reaction time to changes*, and *Platform independence* are less popular, with the latter typically expected by less than 40% of respondents.

RQ6.2: Which are the relations between expectations?

We report the statistically significant relations among benefits in the graph shown in Figure 2.11: the nodes represent the individual benefits, the edges represent a statistically significant relation which is reported as edge label. The layout of the nodes is computed considering the Kendall rank correlation coefficient (KC) (the length of the edge should be as much as possible inversely proportional to the

Table 2.7. Frequency of expectations

Benefit	Freq.	Proportion Estimate	95% C.I.	Fulfillment Rate
Improved documentation	81	77%	(68% , 85%)	68%
Design support	77	73%	(64% , 81%)	78%
Quality of the software	75	71%	(62% , 80%)	49%
Maintenance support	66	63%	(53% , 72%)	52%
Standardization	64	61%	(51% , 70%)	52%
Improved development flexibility	51	49%	(39% , 58%)	45%
Improved productivity	42	40%	(31% , 50%)	45%
Shorter reaction time to changes	41	39%	(30% , 49%)	37%
Platform independence	32	30%	(22% , 40%)	34%

Kendall distance) and additional constraints to improve the readability avoiding the overlaps of nodes and labels.

The benefit expected together ($KC > 0$) are linked by continuous black lines, while the benefits whose expectations tend to exclude each other ($KC < 0$) are linked by dashed red lines, with circles at the ends.

All the significant relations were positive except one, that between *Improved documentation* and *Improved development flexibility*: who expects one of these two benefits tend to not expect the other one.

By observing Figure 2.11, we can note two distinct clusters: the first includes *Improved documentation*, *Design support* and *Maintenance support*. The second one includes *Improved development flexibility*, *Shorter reaction time to changes*, *Platform independence*, *Standardization* and *Improved productivity*. *Quality of the software* appears to be a transversal benefit, connecting the two clusters.

The two cluster contain three maximal cliques[11]: the smallest (left side) cluster correspond to a three-vertexes maximal clique, while the largest one (right side) correspond to a four-vertexes and a three-vertexes cliques that share a node (*Reactivity to changes*).

RQ6.3: Does experience in modeling improves accuracy of benefits achievement forecasts?

The low experienced practitioners group (< 5 years of experience in modeling) is constituted by 50 persons, whereas the high experienced practitioners group (i.e., $\geq$ 5 years of experience in modeling) by 55. Thus, the two groups are balanced.

Applying the Fisher test to the built contingency table, even adopting a looser threshold of 0.1, it is not possible to find any statistically significant difference.

[11]From Wikipedia: in the mathematical area of graph theory, a clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.

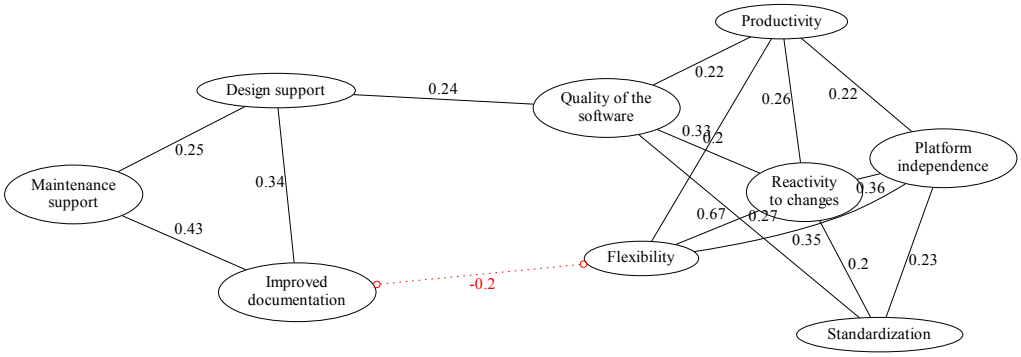

Figure 2.11. Relations among benefits expectations.

Therefore, we conclude that experience does not improve the precision in forecasting the obtainable benefits.

2.5.2 RQ7: benefits achievement

This research question concerns how often the verification of a benefit met the expectation. It is measured as the frequency of verified benefit given the benefit was expected. Results are reported in the rightmost column of Table 2.7 (Fulfillment rate).

Design support has the highest fulfilment rate: 60 respondents out of the 81 who reported to expect it (i.e., 78%) actually achieved the benefit. Also *Documentation improvement* is consistently verified when expected, the same is not true for all the other benefits. *Standardization* and *Maintenance support* are just above the parity (it means are slightly mainly achieved than not achieved, when expected) and all the others are more often not achieved than achieved. *Platform independence* and *Reactivity to changes* have a really low fulfilment rate, representing very often a delusion for practitioners.

Cross-correlations among benefits achievements

Table 2.9 (in appendix) reports the correlation (in terms of Pearson's ϕ) among the different benefit achievements; in bold the statistically significant correlations. We observed one strong correlation (between *Flexibility* and *Reactivity to changes*), 11 moderate correlations, 18 small, and 6 negligible. The strong correlation is a clue indicating a possible common shared underlying construct, as discussed later in the *Threats to validity* section.

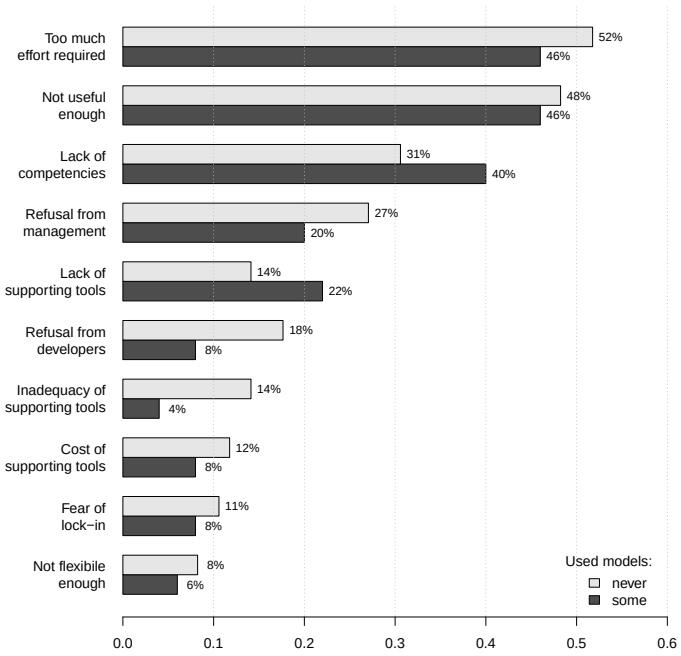

Figure 2.12. Prevalence of problems limiting adoption of modeling.

Factors affecting benefits achievement

After looking at which benefits are concretely achieved by using models, we look into the factors affecting the verification of those benefits: basic modelling vs. MD*, individual key MD* techniques, and additional co-factors.

Overall Table 2.3 (in the first six columns) reports the frequency of achieved advantages among the 105 modelling users. The table also reports the estimated benefit achievement ratio and the relative 95% CI, as percentages. The benefits are sorted from the most likely (*Design*) to the less one (*Platform independence*). According to the likelihood categories defined in section 2.2.5 we can classify two benefits as *very likely*: usefulness for design assessment (*Design support*) and documentation improvement (*Documentation*). In addition, we found that improved comprehension during maintenance (*Maintenance*), higher product quality (*Quality*), and improved standard compliance (*Standardization*) are *likely* benefits. In the *Possible* category fall *Flexibility*, *Productivity*, and *Reactivity to changes*. Eventually, *Platform independence* can be considered as *Unlikely*.

MD* adopters vs. Basic Modellers On the left-hand side of Figure 2.10, we report the benefit achievement ratio separately for users of MD* techniques

and those of basic modelling. From the diagram we can appreciate that for most benefits the difference between more advance users (MD*) and basic users (basic modelling) is limited. The odds ratios of the achievement ratios are reported in the sixth column of table 2.3 aside the p value of the Fisher test. The odds ratios are almost all greater than 1, indicating an improvement of the achievement ratio for MD* adopters, the exception being documentation benefits that appear to be less likely achieved among MD* adopters.

From the test results we can infer that a significant difference exists only for three benefits: *Standardization*, *Productivity*, and *Platform independence* (they are marked with a circle in Figure 2.10). MD* adopters are two and half times more likely to achieve standardization benefits compared to basic modelling users, five and half times more likely to achieve productivity benefits, and five times more likely to achieve platform independence. We observe that the above differences are consistent with the different intended purposes of modelling and MD*: communication the former and code generation or execution the latter. For these three benefits, we classified the likelihood of benefits for MD* adopters; the only benefit that could be classified in a different category was *Platform independence*: considering all modellers it was considered *Unlikely*, while restricting to MD* adopters it becomes *Possible*.

Key MD* techniques More in detail, on the right-hand side of Figure 2.10, we can observe the achieved benefits divided by adopters of code generation, model interpretation, and model transformation, respectively.

Table 2.3 reports (six rightmost columns) the odds ratios of achieving benefit for adopters of specific MD* key techniques vs. non adopters. Code generation induces one significant difference, concerning *Productivity* which is 3.9 times more likely to be achieved as a benefit when the technique is adopted. Most significant differences in benefit achievement are observed when model interpretation is applied: *Flexibility*, *Productivity*, and *Reactivity to changes* benefits are circa four times more likely to be achieved when interpretation is used, in addition *Platform independence* if almost five times more likely. When model transformation is applied, we observe a eight times increment in *Productivity* achievement likelihood, and a four times increment for *Platform independence*.

The most notable spike is observable in the right-hand diagram of Figure 2.10 and concerns *Productivity* benefit achieved by the adopters of model transformation techniques; it corresponds to the odds ratio of eight we described above.

Company size and additional factors We verified whether a correlation exists between the company size categories and the achievement ratio of each individual benefit. The χ^2 test did not reveal any significant relationship, all p values being

greater than 10%.

As a last step of analysis of the achieved benefits we focus on additional factors: toolsmithing, use of UML and UML profiles, and the adoption of DSLs. Table 2.4 reports the benefit achievement odds ratios for the presence of the additional factors vs. their absence.

We can observe that respondents who developed to some extent their tools (toolsmithing) had a significantly higher likelihood of achieving several benefits: four times higher for *Standardization*, *Flexibility*, and *Productivity* benefits; more than five times higher for *Reactivity to changes* and almost ten times higher for *Platform independence*. Apparently the adoption of UML and UML Profiles is not linked to any increased benefit achievement ratio. The adoption of domain specific languages (DSLs) is linked to three times higher likelihood of achieving *Flexibility* and *Productivity* benefits, and four times higher chances to achieve *Platform independence*.

2.5.3 RQ8: problems

As a preliminary step we assess the cross correlation among the different problems. Table 2.10 reports the correlation coefficients (Pearson's ϕ) for each pair of problems; in bold are reported the statistically significant correlations. We observe no strong correlation, three moderate ones, and 14 small ones. Most of the correlations (28) are negligible.

Overall

Table 2.8 reports the frequency of each problem and the corresponding occurrence ratio. According to the 95% confidence interval (columns 4 and 5) we can assign a relevance category conforming to the thresholds defined in section 2.2.5. We observe three relevant potential problems: *Too much effort required*, *Not enough expected usefulness*, and *Lack of competencies*.

Sometimes vs. Never

Figure 2.12 reports the frequency of problems preventing the adoption of models (as reported by respondents never using modelling) or hindering it (respondents using modelling just sometimes) is reported. We remind that participants that use models always were not asked this question.

We looked at the odds ratios of problem occurrence for respondent that never adopt modelling vs. those who sometimes adopt, all the values are close to 1 – between 0.65 and 1.71 – except for *Refusal from developers* and *Inadequacy of supporting tools* having with odds ratio 0.41 and 0.26 respectively. Indicating they are the most likely reasons for not adopting modelling.

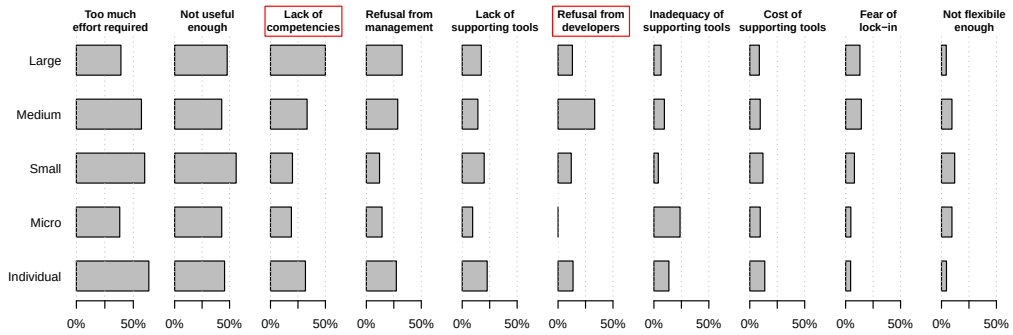

Figure 2.13. Problem occurrence ratio per company size category.

We could not observe any statistically significant difference between the problem occurrence ratio of modellers vs. non modellers.

Basic Modellers vs. MD* adopters

Table 2.8 also reports the frequency and occurrence ratio of each problem among basic modelling adopters vs. MD* technique users. In addition (rightmost two columns) the odds ratio of incurring in a problem for MD* vs. basic modellers is reported together with the relative statistical significance, computed by means of the Fisher test. The only statistically significant difference concerns the *Fear of lock-in*, which appears almost six times more frequent among the MD* techniques adopters. Next to this difference, though not significant, is the one about the *Inadequacy of supporting tools*.

Company size

Figure 2.13 reports the distribution of problems occurrence ratio by company size. We can observe a substantially uniform distribution among the different size classes, with the exception of a few cases. According to the χ^2 test, we identified two problems whose occurrence is significantly related to the company size: *Lack of competencies* and *Refusal from developers*. Concerning the former, small and micro companies are surprisingly reporting this problem less than medium, large, and individual companies. *Refusal from developers* is instead mostly reported in medium sized companies.

	Occurrence ratio			Relevance	Basic		MD*		MD* vs. Basic	
	Freq	Ratio	95% CI		Num	Prop.	Num	Prop.	OR	p
Too much effort required	67	50%	41% .. 58%	Relevant	23	46%	21	60%	1.75	0.27
Not useful enough	64	47%	39% .. 56%	Relevant	26	52%	15	42%	0.70	0.51
Lack of competencies	46	34%	26% .. 43%	Relevant	16	32%	10	28%	0.85	0.81
Refusal from management	33	24%	18% .. 33%	Moderately Relevant	14	28%	9	25%	0.89	1.00
Lack of supporting tools	23	17%	11% .. 25%	Moderately Relevant	9	18%	3	8%	0.43	0.34
Refusal from developers	19	14%	9% .. 21%	Scarcely Relevant	9	18%	6	17%	0.94	1.00
Inadequacy of supporting tools	14	10%	6% .. 17%	Scarcely Relevant	4	8%	8	22%	3.36	0.06
Cost of supporting tools	14	10%	6% .. 17%	Scarcely Relevant	6	12%	4	11%	0.95	1.00
Fear of lock in	13	10%	5% .. 16%	Scarcely Relevant	2	4%	7	20%	5.87	**0.03**
Not flexible enough	10	7%	4% .. 14%	Scarcely Relevant	4	8%	3	8%	1.08	1.00

Table 2.8. Problems encountered preventing adoption of MD*.

2.6 Debriefing session

After the data analysis and interpretation of the results, we conducted a debriefing session with three expert software professionals, which participated in our survey, so as to understand MD* findings that are difficult to interpret.

The experts we asked for clarification cover different features of MD*: (1) is the responsible architect for the design of an in-house MD* solution (in short, a suite for the rapid development of information systems) for a large organization; (2) is the CEO of a company producing a MD* tool for the development of Web applications based on code generation; (3) is the Sales & Marketing Director of a company producing a model driven Web application framework based on run-time execution of models.

Moreover, we took advantage of the availability of such qualified professionals and asked them what they believe is needed to improve the maturity of MD* in the Italian industry.

The outcome of the interviews with the three experts is summarized by the mind map shown in Figure 2.14. The experts are identified by a number in the mind map: the legend located inside the figure explains which expert corresponds to a given number (e.g., the number two identifies the CEO of the company producing a MD* tool for the development of Web applications). In that figure, the four main issues are reported by means of slogans (e.g., *Higher maturity of micro-companies*). Then, for each issue we listed the explanations given by the experts as nodes. The explanations can be supported by one or more experts: e.g., the first explanation of the first issue (i.e, *Small companies can afford simpler, and possibly non very mature, frameworks, ...*) is supported by two different experts while the second (i.e, *More flexible in adopting new processes and technologies*) is supported by just one of the experts (the number two). Only one explanation was supported by all the experts while the others are supported by one or two experts. That should be considered while interpreting the information provided.

2.6.1 Issue 1) Experience in MD* is very low

Low experience in MD* has been attributed to several different causes: mostly the fact that apparently models are used as tools for documentation or analysis and not as artefacts inserted into a MD* approach. According to our experts, the primary cause for this is little popularity in the industry that limits the capability for developers to build "on-the-job experience". One of the professionals also suggested that huge standardization effort, that bring countless notations and techniques together, may result intimidating and actually preventing diffusion of MD* practices. The considerations from the experts strengthened our consideration about the opportunities in the industry and academia.

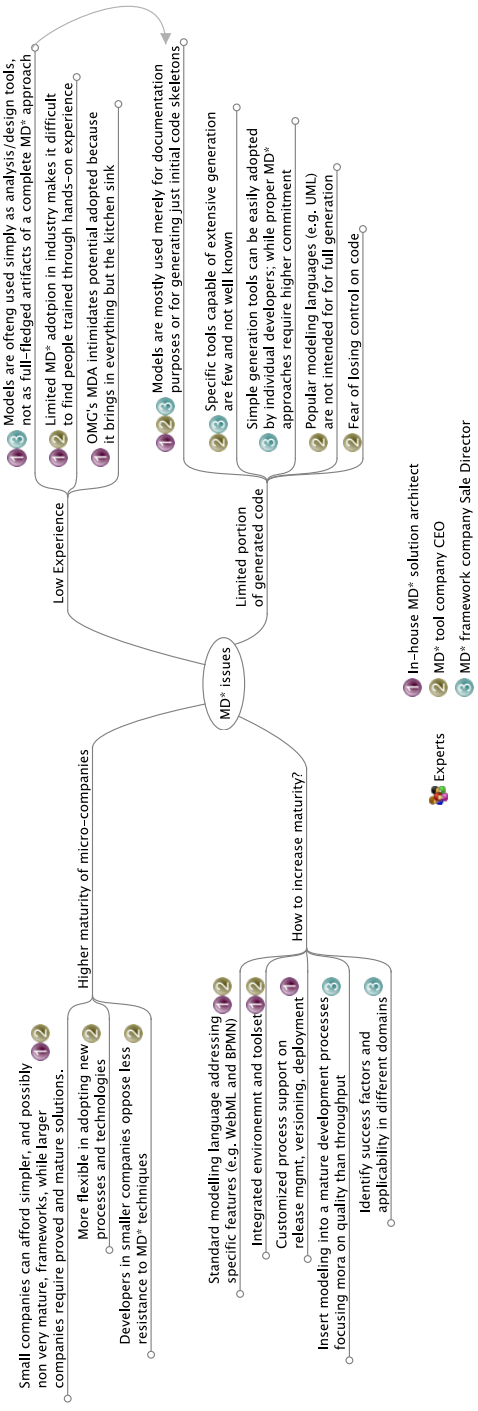

Figure 2.14. Mind map of the experts' opinions

2.6.2 Issue 2) The percentage of code that is generated is often low

Here, returns the supposed use of models barely for documentation purposes or for generating just the initial code skeletons; the typical case consists of generating the code structure from a UML class diagram. In our experts' opinion, the primary cause for this limited use of models lies in the scarcity of appropriate tools and the limited knowledge of the few available. In practice, limited tools are used at individual developer's level because of several factors: it is difficult to get management commitment at team or organization level, common modelling languages (e.g., UML) allow just a limited code generation, and aim for more extensive generation clashes with the fear of losing control over code.

2.6.3 Issue 3) Micro-companies appear to be more mature in MD* than larger companies

Only two of our experts offered an explanation. Small companies can afford to adopt non fully mature solutions, which are not easily accepted in larger companies, and the "small" size allows more flexibility in using new technologies and processes. Moreover, in large companies there is more resistance, by developers, to the introduction of novel techniques and processes than in smaller ones: novelties threaten personal competence niches which are more likely in large companies.

2.6.4 Question) What is needed to improve the maturity and foster the diffusion of MD* in Italy?

The experts mentioned factors in three categories: languages, tools, and processes. Standardized languages are the key to the diffusion of MD* approaches; UML and BPMN are positive examples but are not sufficient because they do not cover all the relevant aspects (e.g. interactions and systems communication). Moreover, MD* usage requires integrated toolsets supporting the full development process. From a process perspective, there is a need for customized processes that include not only the generation but also, release management, versioning, and deployment. As far as management is concerned, a successful application of MD* techniques requires understanding the key success factors, the applicability in different domains and the skills required from developers. From a more general perspective, focus on quantitative aspects of software production does not incentive use of models, which can be exploited when quality is considered. In addition, at a management level, it is important to know which are the success factors for different domains and the skills required to practice MD* techniques.

2.7 Discussion

Benefit achievement likelihood due to simple modelling	Observed significant likelihood increment due to Key MD* techniques: Model transformation	Model interpretation	Code generation	Toolsmithing	DSL	
Very likely						Design
Very likely						Documentation
Likely						Quality
Likely						Maintenance
Likely						Standardization
Possible		✔		✔	✔	Flexibility
Possible	✔	✔	✔	✔	✔	Productivity
Possible		✔		✔		Reactivity to changes
Unlikely	✔	✔		✔	✔	Platform independence

Figure 2.15. Achievable benefits with Modelling and MD* techniques adoption effects.

General diffusion The first result we obtained from the survey is that of a large **diffusion** of modelling practices (68% of respondents) and a relatively ample diffusion of MD* techniques (48% of the adopters of modeling and 32% of the entire valid sample). Considering our plain relevance criteria, we can classify modeling as a highly relevant technique in the industrial context; while MD* can be considered as relevant. This information is important per-se for us researchers: it means that research conducted in this context has the potential to yield a significant impact on practitioners.

When looking at how the adoption is distributed with respect to company size, we observe a bimodal shape both for modelling and MD*: medium-large companies are more keen to adopt modelling and MD*, small companies are less prone to these practices, and micro and individual companies are similar to large ones in this respect.

Specific techniques diffusion As far as key MD* techniques are concerned, almost all adopters of MD* do apply code generation, one third apply model interpretation, and one fifth use model transformations. The adoption rate of code

generation by micro companies[12] is significantly higher than other companies while the two latter techniques are largely adopted by micro companies only. The large diffusion of those techniques in micro companies is in stark contrast to the very low adoption in small companies.

Our explanation for the above facts is that model interpretation and transformation are relatively novel techniques, at least more advanced than code generation which is well-known and more used in the industry. As a consequence, their adoption brings risks: apparently only micro companies, and to a much lesser extent medium-large ones, are willing to take them. Micro companies are possibly driven by the competition to stay in the market, medium-large ones perhaps believe in the advanced techniques as competitive advantages. Moreover, micro companies can afford to adopt, more easily, not fully mature solutions — nowadays, provided in the market for Model transformation and interpretation — which are not easily accepted in larger companies. In addition, the "micro" size allows them more flexibility in using new technologies and processes (one possible co-cause for that could be the larger freedom developers have in micro companies). Finally, probably, in large companies there is more resistance, by developers, to the introduction of novel techniques and processes than in smaller ones.

Benefits achievement Another important finding concerns the likelihood of **benefits** achievement. The adoption of modelling makes improved design and documentation benefits very likely to be achieved (see Table 2.3), while maintenance, quality, and standardization are simply likely.

Figure 2.15 shows the the relationship between MD* techniques and the benefits: each checkmark indicates that the adoption of the technique in the column causes a statistically significant improvement in the likelihood of achieving the benefit in that row.

In practice the four topmost rows in Figure 2.15 represent the most commonly achievable benefits. *Support in design definition*, *Improved documentation*, easier *Maintenance*, and higher *Quality* are obtained through the simple adoption of modelling. We could not find statistically significant evidence of any MD* effect on their achievement because, due to the design of our survey, we could not compare modellers vs. non modellers. (Table 2.3).

MD* specific techniques play a significant role for the remaining five benefits investigated in our study. The odds of achieving *Flexibility* is almost four times higher when model interpretation is adopted. When it comes to *Productivity* all the three MD* techniques increase the likelihood obtaining an improvement, in particular model transformation may increase the odds by eight times. *Reactivity to*

[12] $2 \leq \text{size} \leq 10$

changes is easier to achieve when model interpretation is adopted, while the chances of achieving *Platform independence* are increased by applying model transformation or model interpretation.

Moreover, Figure 2.15 shows how other two techniques – toolsmithing and DSLs –, usually associated with MD* practices, can play a significant role. The development of own tools (toolsmithing) is a significant enabler for all bottom five benefits in the figure (*Flexibility*, *Productivity*, *Reactivity to changes*, and *Platform independence*). Particularly relevant is the contribution of toolsmithing to *Platform independence*: an Odds ratio of 10 is very high and suggests that projects having platform independence among their priorities should seriously consider building their own tools. Unfortunately, we have no information about the effort required to realize those tools. More evidence is needed to drive the "make or buy" decision. Finally, the use of Domain Specific Languages increases the odds of achieving *Flexibility*, *Productivity*, and *Platform independence*, by three to four times.

We can interpret statistical significance as a causal relationship, which represents the empirical basis for pragmatic decision making. Under such perspective, Figure 2.15 illustrates the factors that can play the role of the deal breaker in achieving a given benefit, on the basis of the collected empirical evidence.

In practice, Figure 2.15 is an attempt to synthesise a piece of evidence that can be leveraged by practitioners. For instance, if the goal for a project is *Productivity*, all the three MD* specific techniques – code generation, model interpretation and model transformation – can help; while if we aim at achieving *Reactivity to changes* only model interpretation and toolsmithing can help.

We are not claiming the solutions derivable by Figure 2.15 are the only possible ones: they are combinations that proved statistically significant in our sample. As such, they represent the starting point in finding a customized solution for a goal or set of goals.

Another insight we get from a overall glance at Figure 2.15 is that MD* and other techniques play an important role where the simple modelling is weaker and vice-versa. From this picture we confirm the impression that simple modelling and MD* are two complementary sets of techniques.

Problems If we look at the **problems** restraining from the adoption of modelling, as reported by respondents, we observe that most problems are cited less often by participants that never use modelling than those who use it sometimes (see Figure 2.12). The possible explanation for this difference is that potential problems and risks are considered more often when development teams has to repeatedly balance pros and cons for using modelling or MD*.

The notable exception to the above trend is represent by the *Lack of competencies* and *Lack of supporting tools*. Such problems appear to be the main show-stoppers

preventing altogether the adoption of modelling and MD*.

The *Fear of lock-in* is a problem which seem to affect a lot more MD* practitioners than adopters of simple modelling. This could derive from the lack of affirmed standards in MD* while in modelling UML seems to be widely used. This lack results into both poor options for replacing tools by equivalent alternatives and problems in building heterogenous tool-chains.

We believe that the findings of our survey, and in particular, these two above problems (*Lack of competencies* and *Lack of supporting tools*) deserve attention from Italian industries and universities. The former should invest more in research, tools building (software modelling and MD* tools are needed) and training (experts in MD* are needed), and the latter should produce more experts in modeling and model driven techniques. This strongly suggests to improve university curricula with specific courses dealing with topics related to software modelling, and more specifically with code generation, model execution, and model transformation. Most of the times, students are trained to build new systems using traditional processes and only in the better case the foundation of MD* are explained in software engineering courses (e.g., this is the case in the Università di Genova — Italy). While it is our opinion that they should focus more on modelling and model driven techniques (in particular in automatic code generation, given that, it is the most used in the industry). On the university side, the *Lack of supporting tools* and dissatisfaction about them (Figure 2.12) should be a prompt to produce new prototypes and experiment more in this direction. On the industrial side, we can infer a huge market opportunity for modeling and MD* tools. Moreover, investments in this market could obtain large returns especially for large companies. And possibly, companies and universities should collaborate to make better tools.

2.8 Threats to validity

Industrial surveys are intrinsecly difficult to realize and common threats to validity listed in [Torchiano and Ricca, 2013].

We analyse the potential threats to the validity of our study according to the four categories suggested in [Wohlin et al., 2000]. In general on-line surveys are considered to have lower internal validity and stronger external validity in respect to other means of empirical investigations as case-studies or experiments [Punter et al., 2003].

Construct validity threats concern the relationship between theory and observation. They are mainly related to the measurements performed in the study. In particular there are two distinct issues: (i) whether we measured the usage of techniques in the right way (measurement instrument and process) and, (ii), whether we

selected the right attributes to represent the construct of technology usage (measured attribute).

Concerning the measurement instrument – a personal opinion survey – the items definitions and the scales used to code the answers are key factors, and they can potentially influence our results because of the difficulty respondents could have encountered in either understanding or (mis)interpreting the items. We paid particular attention to mitigate such threats:

- The questionnaire was designed using a standard approach, trying to avoid as much as possible ambiguous questions [Kitchenham and Pfleeger, 2008].
- We strived to formulate the items in a simple and straightforward style (see Table 3.1)
- We inserted the meaning of relevant terms (e.g., model, model execution, MD*) in the questionnaire (directly in the questions or as footnotes).
- We guaranteed assistance by phone and e-mail to respondents, to support them in case of unclear questions.
- We conducted a pre-test of the instrument, by means of three pilot studies with industrial professionals, to check that the questions were understandable, before putting on-line the questionnaire.
- In particular the lists of advantages (question Dev11) and problems (question Dev09) proposed to the respondents could have been incomplete. In the first case, we opted for a closed question (see Figure 2.2) on the basis of the expert practitioners judgement in the pilot. On the contrary, in the second case, since the range of problems is potentially very large, we opted for a semi-open question: we provided a set of predefined options but let the respondent free to add others (see Figure 2.2). Considering the answers of Dev9, we can confirm that the list of proposed problems was quite complete, since respondents sparingly used the "free" option.

As far as the measured attributes are concerned, we identified the MD* usage construct with the adoption of three key techniques: code generation, model transformation, and model interpretation. We are confident that they represent the main features of MD*, or at least a largely shared view of MD*, although we cannot exclude there exists some community with a different perspective on MD*.

In addition from the correlation analysis, we found a strong correlation between *Reactivity to changes* and *Flexibility*; this may indicate the possibility of a single construct underlying the two measures. We explored the possibility of removing one of the two, but eventually we preferred to retain all the information and keep both

at the expense of a small additional complexity of the study. We believe that this decision does not invalidate the conclusions of our study.

Internal validity threats concern confounding factors that may affect the outcome of our results. In general, it is hard to control these factors. It is well-known that, a survey being an unsupervised study, the level of control is very low.

Internal validity is mainly threatened by coverage issues:

- We incurred in a possible selection bias due to the self-exclusion of participants not interested in modelling. Self-exclusion is a well-known problem especially in Internet surveys advertised by means of mailing lists and groups. The possible threat consists in an over estimation of the proportion of respondents who declared interest in modelling and therefore of the overall relevance of modelling and MD* in the Italian industry. The main impact of this issue would be on the answer to RQ1, the findings relative to the other research questions should not be significantly affected.

  We tried to mitigate this threat by presenting the request as a study on software development without emphasizing the modelling aspect and addressing it to a varied population. Though we cannot definitely rule out the threat, there are two aspects in the collected data that make us at least confident that the magnitude of the threat is limited. First, a significant number of valid respondents never used modeling therefore a portion of the sampled population although not interested in modelling did filled in the survey anyway. Second, among the incomplete answers – respondents who started the questionnaire but did not complete it and were thus discarded from analysis – (26 occurrences), 10 respondents did anyway provide a response to question DEV08: all of them affirmed to use models *sometimes* (8 respondents) or *always* (2 respondents); if in a conservative move we ascribe the remaining 16 to the group of respondents *never* using modelling, the picture we obtain considering all the 181 (155 complete + 26 incomplete) responses is not significantly different than the complete responses alone.

- Another threat derives from the possible "foreign units" in the sample: the target population of our study consisted of development teams, it is possible that the questions were answered by a responded without the required knowledge (e.g., the secretary of the IT manager). We addressed this concern in the protocol: we explicitly required the questionnaire to be filled in by technical personnel involved in the development. Even in the case of a knowledgeable respondent, (s)he could be unaware of some details; this is more likely if the team is very large [Kitchenham and Pfleeger, 2008].

- Finally, the sampling procedure made possible to select duplicate units: two different members of the same development team could have answered our

questionnaire. We addressed this threat by means of a post-survey validation: we found that the respondents from the same company actually worked in distinct business units and belonged to distinct teams.

Conclusion validity threats concern the possibility to derive illegitimate conclusions from the observations. When hypothesis testing was used to compare two or more populations, we adopted non-parametric tests (Kruskal-Wallis, Fisher exact test and Mann-Whitney), that can be used without specific assumptions (e.g., without checking data normality). Similarly, we used the proportion test to determine the confidence interval for relevance of modelling in general.

External validity threats concern the extent to which our findings can be generalized. For our survey, we used the Commerce Chamber database to render our sample as representative as possible. Had we used only this sampling source, we could have performed a stratified sampling, using strata based on company size as was performed e.g. in [Egorova et al., 2010]. However, we decided to integrat that sample using a non-probabilistic sampling schema (the same was done in [Nugroho and Chaudron, 2008]). As a result, the solution we selected is cost-effective and allowed us to obtain a sample large enough to achieve a reasonable number of adopters of all the techniques. This should be considered interpreting the results we obtained: even if the demographics of our sample is quite diverse, the generalization of our results to the entire population may not be appropriate. Moreover, given the sampling strategy we adopted, we cannot calculate the response rate (this problem is common in software engineering surveys [Kitchenham and Pfleeger, 2008]). We are also aware that the size of our sample is not large enough for generalization purposes (of course, further data points will be highly desired to better generalize our findings). However, that size is similar to other industrial surveys conducted on different software engineering subjects (e.g., [Hauge, 2007, Hutchinson et al., 2011b, Jelitshka et al., 2007, Li et al., 2008, Torchiano et al., 2011a]).

2.9 Related work

Literature reports some anecdotal evidence collected through case-studies (e.g., [MacDonald et al., 2005]), while rigorous empirical studies evaluating software modelling and MD* are quite rare. Carver et al. [Carver et al., 2011] performed a literature review considering the most common venues where articles related to MD* are published. They noted that the 73% of the papers they considered didn't contain any form of validation, consequently they affirm that the rigor of empirically validated research in software modeling is rather weak and the community need to focus more on this aspect. In particular, as stated in [Mohagheghi and Dehlen, 2010], there are a few reports on the advantages of applying MDE in industry, thus

more empirical studies are needed to strengthen the evidence. Also van Deursen et al. [van Deursen et al., 2007] report on the importance of having more empirical studies, in particular about the improvements produced by MD* adoption on maintainability costs.

In the rest of this section, we focus on empirical studies about issues, challenges and benefit of modelling and MD* adoption. Coherently with the stance adopted in our survey, we decided to avoid a clear partitioning between software modelling and MD*, also given that the two aspects are often interwoven.

We grouped the related studies as much as possible considering their category: literature reviews, surveys, case studies, and experience reports.

2.9.1 Literature reviews

Budgen et al. [Budgen et al., 2011] conducted a systematic literature review on UML. Authors underline the necessity of more empirical studies about the adoption of UML. Most of the empirical studies are laboratory experiments while more field studies are needed.

Mohagheghi and Dehlen in their work [Mohagheghi and Dehlen, 2010] introduce a literature review of empirical studies (and more in general of industrial experiences), from 2000 to 2007, about MDE in industry. The goal is evaluating MDE benefits and possible limitations. They selected 25 papers and their main conclusions are: (i) MDE is applied in a wide range of domains, (ii) MDE can lead to various benefits (i.e., higher productivity and improved communication), (iii) MDE is not considered mature enough and there are no appropriate tool chains, and (iv) quantitative evidence was found in one paper only, about productivity gains. Our findings are consistent with theirs, especially with items (ii) and (iii).

2.9.2 Surveys

Forward et al. in their work [Forward et al., 2010] analyses the results of a survey with 113 software practitioners (about two-thirds were from Canada or the USA) on the perception of software modeling. They mainly investigate how, when and why software developers use (or not) models and which notations and tools are adopted. In their sample, modelling is performed at least sometimes by over 95% of participants (in our case the modellers are 68%). Similarly to us, they try to answer the following research question:"Why do some developers prefer not to model?" (similar to our RQ3). They report that the biggest problem is the synchronization between models and code (models become out of date with code). We do not have evidence of this problem. Other problems are the quality of the generated code and issues with the modelling tools (e.g., too expensive, "heavyweight" and difficult to use). Our results are in line with this latter perception. A portion of

our sample think that modelling tools represent a limitation for MD* and more in general for software modelling. The main findings reported in their conclusion are: (i) developers consider models in a broader sense (i.e., not only UML models but also textual DSL models), (ii) UML is the predominant modelling notation but is often used informally, (iii) modelling tools are mainly used for documentation (a fact that could explain the large percentage of documentation benefits reported in Figure 2.10), and (iv) it is uncommon that models are used for generating code.

Davies et al. in [Davies et al., 2006] report the result of a survey conducted in Australia on the status of conceptual modelling that has received 312 responses. This study aims to determine the actual modelling practice, giving a answer to the research question:"how do practitioners actually use conceptual modelling in practice?" that is specified by three sub-questions: (i)"which are the tools and techniques used for conceptual modelling?" (ii)"what is the purpose of modelling?", and (iii) "what are the major problems and benefits specific to conceptual modelling?" The last sub-question is similar to ours RQ2 and RQ3, but from a different perspective. They have identified problems and benefits in the usage of conceptual modelling by means of textual analysis of data relative to problems and perceived key success factors. Concerning the last sub-questions they report which are the factors influencing the continued use. The major key factor is relative advantage/usefulness, other factors (in order of relative importance) are: communication to/from stakeholders, internal knowledge of techniques, user expectations management, understanding the model integration into the business, tool/software deficiencies. The first factor corresponds to our finding of a commonly achieved benefit (documentation improvement), the last factor corresponds to one of the problem we found limiting adoption of modelling (inadequacy of supporting tools).

Hutchinson et al. in [Hutchinson et al., 2011b] report the results of an empirical study on the assessment of MDE in industry. Their work has two goals: identify the reasons of success or failure of MDE and understand how MDE is actually applied in industry. They employed three forms of investigation: questionnaires, interviews, and on site observations, having as target practitioners, MDE professionals and companies practising MDE respectively. The questionnaire has received over 250 responses from professionals (the most of them are working in Europe). Some of the reported findings are: (i) about two-thirds of the respondents believe that using MDE is advantageous in terms of productivity, maintability and portability, (ii) the majority of respondents use UML as modelling language, and a good number use in-house developed DSLs, (iii) almost three quarters of respondents think that an extra training is necessary to use MDE, (iv) the majority of respondents agree that code generation is an important aspect of MDE productivity gain, and (v) a little less than half of the respondents think that MDE tools are too expensive. We observed similar perceptions in our survey except for the issue of extra-training which was not considered in our survey, however we observed that the lack of competencies is

one of the problems most frequently reported by companies. Differently from the results of their survey, the cost of supporting tools is seen as a problem only by a small proportion of respondents in our sample.

Nugroho and Chaudron in their work [Nugroho and Chaudron, 2008] analyses the results of a survey about the UML usage and its perceived impact on quality and productivity with 80 professional software engineers. The findings reveals that: (1) the majority of the respondents agreed that a model should describe parts of a system that are more critical and complex, instead of specifying all parts of a system equally; (2) incompleteness in UML models is associated to implementation problems (it brings to deviations in the implementation) and, (3) using UML impacts productivity in analysis, design and implementation but not in maintenance. This last finding seems partially in contrast with our results where easier maintenance is one of the benefits associated with modelling. Similarly to our survey, their findings reveal problems associated to the usage of tools: the features in current UML CASE tools that should help maintaining aligned code and design, e.g., reverse engineering and round-trip engineering, are not yet mature.

2.9.3 Case studies

The work of Hutchinson, Rouncefield and Whittle [Hutchinson et al., 2011a] focuses on the deployment of MDE in industry. It illustrates three industrial case studies in three different business contexts (printer, car manufacturing and telecommunication companies) and identifies some lessons learned. In particular, the importance of complex organizational, managerial and social factors in the success or failure of the MDE deployment. The authors report some organizational factors that can affect the success or the failure of MDE deployment. The factors that can affect it positively are: (i) a progressive and iterative approach, (ii) user motivation in the MDE approach, (iii) an organizational willingness in integrating MDE in the whole organization, and (iv) having a clear business focus (where MDE is adopted as a solution for new projects). Instead, factors that can affect it negatively are: (i) the decision of adopting MDE being taken by IT managers, in top-down fashion and implemented "all at once", (ii) MDE being imposed on the developers, and (iii) an inflexible organization with a lack of integration of MDE in previous processes. The only common aspect with the work proposed in [Hutchinson et al., 2011a] concerns the motivation of developers. The corresponding finding lies in the problems reported in Figure 2.12 (refusal from developers and refusal from management).

Mohagheghi et al. [Mohagheghi et al., 2012] interviewed – using convenience sampling – developers from four companies involved in an initiative called MODELPLEX. They examined the factors affecting adoption of MDE. Regarding usefulness they found uncertain results: most participants recognize the usefulness of models but they are not sure about the impact on the quality of the final product or

the effects on productivity. MDE is perceived as not simple: its complexity makes it viable for engineers but not for non technical people. This finding is confirmed by our results. They show that only in a few cases business experts are involved during modelling tasks. Regarding compatibility with the existing development process the companies complained about the lack of standards and the consequent lock-in effect. All interviewed companies reported some problems in integrating their existing approaches with MDE. Tools could have been part of their problems, them being not considered satisfying by a part of the sample. In particular, some participants expressed several concerns about the scalability of the MDE approach to large projects. Advantages reported are limited to the usefulness for documentation and communication purposes. Major reasons preventing adoption of MDE are the immaturity of tools and processes as well as the lack of competencies. Such latter conclusions are largely consistent with our findings.

2.9.4 Experience reports

Heijstek et al. [Heijstek and Chaudron, 2009] study the impact of MDD on a large scale industrial project and the main features of a large scale industrial MDD project. They produced an experience report using, as sources of information, data from the Subversion repository and semi-structured interviews with team members. About the impact of MDD, the conclusions are that: almost two-thirds of the total effort is spent on developing models and that the team members report an increase in productivity, besides a perception of improvement of the overall quality and a reduction of complexity. The authors confirmed the increase of quality by counting the average number of defects w.r.t. the average number of defects found in similarly sized projects in which MDD was not used. Their findings – including the improvement of the final product – have been observed also in our survey. While we do not have data about the effort spent on realizing the models, many participants considered that effort to be too big and therefore affecting their decision to adopt modelling. They identified two typical features of large scale MDD projects: (i) high average complexity per diagram, (ii) activity diagrams and class diagrams are the more used UML diagrams.

Mohagheghi et al. [Mohagheghi et al., 2009] report a list of challenges and success criteria of MDE adoption. They, summarize them after having conducted two real projects in large organizations. The most important challenge is the definition of a MDE environment, that require the company: (i) to develop a communication language for technical and domain experts by means of UML profiles and/or metamodels and (ii) to select and integrate tools for building, transforming, storing, reusing and composing models. This last point is particularly difficult to reach due to the lack of such a tool-chain on the market. Such finding is consistent with our results. They also report that training is a major challenge (same problem stated

in [Hutchinson et al., 2011b]). In addition, they see advantages in the gradual introduction of MDE in the industry and in the creation of expert teams to support and create tools.

2.10 Summary and future work

In this chapter we presented some results from a survey performed to investigate: (1) what is the relevance of software modelling and MD* in the Italian industry, (2) the way software modelling and MD* are applied, and (3) the motivations on which adoption is chosen or refused. Here we summarize the findings and discuss directions for future work.

G1) Relevance

We found that the practice of producing models is quite widespread (68% of the entire sample), while the proportion of development teams using MD* techniques is smaller (48% of the adopters of modeling) but still noteworthy. Among the MD* techniques, the most used is code generation (almost all adopters of MD* do apply code generation). Considering our relevance criteria, we can classify modelling as a *highly relevant* technique in the Italian industrial context while MD* can be considered as *relevant*. This result implies that any research gain in this field has the potential of a large return on practice.

G2) Processes, Languages and Tools

Apparently, many companies use models to capture an high level view of the system and for documentation purposes. A possible cause of this reality is that developers are not enough educated in model driven techniques.

UML is largely more used than DSLs.

Over the 50% of companies do not adopt any of the techniques considered (code generation, model interpretation and model transformations). Among this techniques the one being most widespread is code generation.

Models are never created exclusively by business experts but instead technical figures seem to be always included in the process.

G3) Motivations

More relevant benefits such as: *Support in design definition*, *Improved documentation*, easier *Maintenance*, and higher *Quality* seem to be obtained when simple models are used and no further improvement is observed with MD* adoption. On the other hand, MD* plays a significant role for *Productivity*, *Platform independence*

and *Standardization*. The complete set of empirically backed causal relationships (technique adopted → benefit) observed in our study is shown in Figure 2.15. Such figure is an attempt to synthesise pieces of evidence that can be used by practitioners to decide whether to invest on a specific technique, given the desired benefits.

The main problems mentioned by adopters of modelling mimic the typical anecdotal ones: models require too much effort to be produced and often they are not useful enough. Instead, the problems preventing altogether the adoption of both modelling and MD* seems to be related to a mix of technological and human factors, that is lack of supporting tools and lack of competencies.

Future work

Such findings can originate suggestions useful for both Italian companies and universities. The former should invest more in research and tools building in order to address the lack of supporting tools (and in general tools inadequacy) reported by the respondents of our survey. The latter should produce more experts in modelling and model driven techniques so to raise the level of expertise and satisfy today's industrial needs.

As future work, we would like to compare the level of adoption of modelling and MD* in Italian companies to the situation in other countries by replicating this study in other nations. In particular, we are interested to understand whether the companies in other nations have the same problems that the Italian have and whether (or not) they achieve the same benefits using modelling and MD*.

Detailed tables

We report here the cross correlations among the benefits in Table 2.9 and this among the problems in Table 2.10.

Acknowledgements

We would like to thank the people who took part in our survey, the organizers of CodeMotion for advertising the survey, and the experts which we interviewed and discussed our findings with: Paolo Arvati from CSI Piemonte, Stefano Butti from WebRatio and Paolo Predonzani from ManyDesigns

	Design	Documentation	Maintenance	Quality	Standardization	Flexibility	Productivity	Reactivity to changes
Documentation	**0.38**							
Maintenance	**0.33**	**0.21**						
Quality	**0.23**	-0.08	**0.35**					
Standardization	0.07	0.08	0.16	**0.22**				
Flexibility	0.18	-0.13	**0.24**	**0.39**	0.14			
Productivity	0.17	-0.06	0.12	**0.37**	**0.26**	**0.42**		
Reactivity to changes	0.02	-0.12	**0.24**	**0.35**	**0.28**	**0.60**	**0.39**	
Platform independence	0.17	0.10	**0.21**	0.17	**0.25**	**0.36**	**0.38**	**0.36**

Table 2.9. Benefits achievement correlation.

	Not useful enough	Too much effort required	Lack of supporting tools	Inadequacy of supporting tools	Lack of competencies	Fear of lock in	Refusal from developers	Refusal from management	Cost of supporting tools
Too much effort required	0.01								
Lack of supporting tools	0.04	-0.02							
Inadequacy of supporting tools	0.07	0.10	0.04						
Lack of competencies	0.10	-0.12	**0.21**	0.06					
Fear of lock in	0.04	0.08	-0.15	-0.03	0.03				
Refusal from developers	0.09	0.15	-0.01	0.07	0.11	0.08			
Refusal from management	**0.18**	0.02	0.15	0.15	**0.32**	-0.01	**0.22**		
Cost of supporting tools	0.02	0.15	0.04	**0.36**	-0.04	**0.38**	0.07	0.03	
Not flexibile enough	0.01	0.12	0.02	0.09	-0.08	**0.20**	0.13	0.04	**0.18**

Table 2.10. Correlation of potential problems.

Chapter 3

Modeling adoption in a small company: the Trim case-study

In this Chapter we investigate first time adoption of MDD in a small company. Results of this research were published in [Tomassetti et al., 2013a]. It was a joint work realized with Marco Torchiano, and Lorenzo Bazzani.

Results of the work presented in this Chapter contributes to answer the thesis research question **RQ A.4**.

This research presents the knowledge and experience acquired trough the process of establishing MDD practices within a small Italian company. Special attention has been devoted to project constraints, perceived risks, and relative mitigation strategies. Moreover the study evaluates how the introduction of the MDD approach was received by different stakeholders. In particular a structured questionnaire was the instrument used to reveal and collect the perceptions by different roles involved in the MDD adoption process. The case study considered development of applications conforming to a prescriptive architectural framework, which addresses a complex multi-tier architecture; the solution aims at describing component composition while avoiding both repeating tasks and writing awkward configurations.

3.1 Introduction

The MDD family of approaches has been widely adopted in the industry with different gradations and we have reports of the experience acquired in large organizations (e.g., [Baker et al., 2005, Hutchinson et al., 2011a]). As far as small companies are concerned little empirical or even just anecdotal evidence is available in the literature.

Our goal is to investigate the reasons for success or failure for MDD adoption in small industrial contexts. In particular we focused our attention on two aspects: (i)

the potential risks and the associated mitigation strategies, (ii) the overall perception by the different stakeholders. In this work we are specifically considering the initial reception and the effects of deploying for the first time an MDD solution in a small company.

As far as the risks about adopting MDD are concerned, from a single case study it is not possible to derive a statistically based picture, although by relying on the managers experience we were able to get a clear idea of the perceived risks. It is important to focus on the perceived risks because they, even though possibly not empirically grounded, are typically responsible for early rejection of MDD approaches. Moreover while the promises of MDD are known, we could not find in the literature, despite of our best effort, an analysis of risks and motivations that cause companies to fail or abandon adopting MDD. A report about the perceived risks emerged while designing and implementing a MDD solution and the lesson learned about coping with them, apparently represent an important contribution to the field.

Furthermore the success of MDD practices adoption depends significantly on how different roles subjectively appreciate them, because participants' commitment represent a key factor. A questionnaire (see Table 3.1) administered to all the participants to the project allowed us to assess the appreciation and perception from different stakeholders' perspective.

The research we present here was conducted in the context of a technology transfer initiative involving the Softeng group at Politecnico di Torino and a small ICT company (Trim srl). The main goal of the collaboration was to design and implement a MDD solution based on a Domain Specific Language (DSL) and the supporting tools. In particular the DSL is used to describe the structure of enterprise applications, realized in conformance with a prescriptive reference architecture composed by seven different layers. The DSL allows designing how the different components are linked together, in order to realize the application, and defining the data structures exchanged between the layers. By using instruments from the Eclipse Modeling project we were able to realize the supporting tools in a short period and with a small effort (circa 64K lines of Java code developed in 5 months of full-time work by one of the academical authors of the paper). Moreover, we realized an integrated environment by combining such tools with the Eclipse Java IDE used at the company.

The main contributions of the chapter are: (i) an examination of the main risks perceived by SMEs in adopting an MDD solution, (ii) an analysis of the risk mitigation strategies adopted, (iii) an investigation of the acceptance of the MDD solution by different stakeholders.

In the remainder of the paper Section 3.2 first presents the project considered for the case study, then Section 3.3 gives an overview of the devised MDD solution, Section 3.4 presents a risk management perspective on the MDD adoption, Section 3.5 discusses how different stakeholders accepted the new approach, Section 3.6

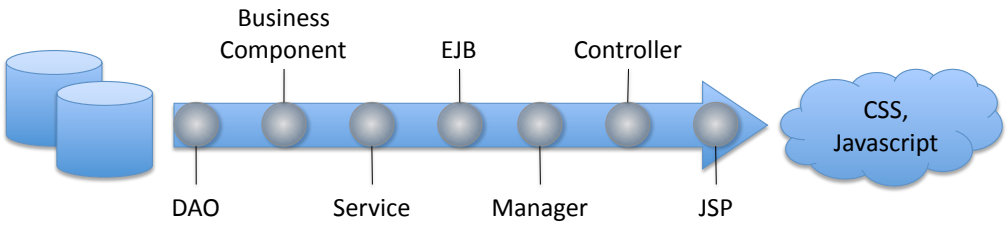

Figure 3.1. Layers of the case study architecture

summarizes and compares related works, and finally in Section 3.7 we draw the conclusions.

3.2 Case Study background

Today common enterprise applications comprise many layers [Singh et al., 2002]. At the bottom we almost always find a relational database and the code needed to deal with it (Data Access Objects or DAOs). At the top lays the presentation layer (i.e., JSPs when the chosen technology is Java). In the middle we can find a variable number of layers composing elementary services, dealing with transactions, playing the Model or the Controller roles in the Model-View-Controller (MVC) pattern. The prescriptive architecture (named Financial Value Chain or FVC) involved in this work is no exception being composed by seven different layers. Those layers are reported in Figure 3.1.

Each layer brings into the project a different technology. In addition specific technologies are required to let the layers communicate while preserving abstraction. A common choice in a Java-based technology stack is to adopt Spring to manage dependency injection. Knowledge of several different technologies is required to deal with a such varying spectrum of libraries and frameworks. In a small company, counting a small number of developers, it could be difficult to either find or build this knowledge. There is a certain amount of code and configuration that does not implement any business logic but it is required just to realize the technological infrastructure, e.g., the composition of different elements and the communication between layers. We refer to it as architectural code as opposed to the business logic code. The architectural code and the related configuration often happens to be repetitive and as consequence error-prone [Sutcliffe et al., 1999].

3.2.1 Motivations for MDD

In the context of the project the company had to use a complex multi-layer architecture. Development for this architecture involved a large amount of repetitive and error prone tasks which demovatived the developers, moreover only developers with a large skillset were able to work on this architecture. For these reasons we decided to encapsulate technological details related to the architectural code in the transformations. In this way developers could focus on the business logic.

3.2.2 Project constraints

The design and implementation of the MDD solution we carried on was tailored for the specific needs of a company, therefore it had to confront with a set of different constraints, the most relevant being: i) since the software development project, where the approach had to be put in trial, involved several companies, it required the integration of components developed by 3-rd parties; ii) the client imposed quite strict rules for source code and configuration files that the generation phase had to adhere to; iii) the company management preferred the use of protected region with respect to other technical solutions.

Concerning the protected regions, the preference was mainly due to the fact it allowed to obtain generated code very similar to manually-written code. Such similarity, in case the MDD solution be abandoned, would make the fallback option to traditional development techniques easier and less expensive. An alternative to protected regions was to adopt specific strategies to separate generated code from manually written code (e.g., aspect oriented programming). These solutions while offering advantages would produce code different from the one produced by the standard development method. While for mature adopters of MDD the usage of protected-region could be considered an anti-pattern, in this case it was considered to be a pragmatic compromise increasing the confidence of the adopters.

3.2.3 Perceived risks

Due to the lack of previous experience in MDD at the company, we ended up debating a set of perceived risks, that are quite common among new MDD adopters. We conducted an open discussion session with the managers of the project aimed to elicit such risks; we wrote and refined a list of risks and eventually we validate the list back with the managers. As a result we could summarize four main risks:

R1) Tool rigidity: it is very likely that new development's best practices appear in the project lifetime, the tool could not be able to evolve to address them.

R2) Lack of developer adoption: the tool and the change in the development process could not be accepted by developers. They could feel like they are losing control or their skills are considered less valuable because of the MDD adoption.

R3) Solution lock-in: the company, at its first MDD experience, in case of failure could not be able to switch back to the previous (MDD-less) development process for the whole project or for the development of a single component with a limited effort.

R4) Application evolutionary inertia: the context of the application is doomed to rapidly evolve, both in terms of development technologies and as new domain and application requirements; while the company knows how to cope with such evolution using standard development, the fear is that MDD could make it more difficult. The request is that adopting the tool should not make reaction times longer but equal or possibly shorter than conventional development.

3.2.4 Scope of the solution

For the development of the lowest layer of the prescriptive architecture (FVC) an Object-Relational Mapping (ORM) was already in use. In particular the ORM used was iBatis and it was used with a companion tool, iBator, which analyzed the database schema and generated DAOs and configuration for iBatis.

The results obtained using iBatis were satisfactory so we decided to design an MDD solution that did not involve the database and DAO layers but instead was able to integrate with those layers as implemented using iBatis and iBator. Being the first MDD adoption trial at the company we wanted to focus only on the most problematic layers, where it was easier to obtain a significant improvement.

The company preferred to not adopt MDD for the presentation layer because they wanted their developers to maintain full control on this particular aspect of the application. Therefore the MDD solution was designed to cover five out of the seven layers of the FVC architecture.

The design of the MDD solution was driven first of all by pragmatism: while in an environment with experience in MDD usage we would have suggested a different approach in this case we had to look for a compromise between best practices and risk perceived. For this reason on one side we chose to focus on a subset of the layers and in particular on the ones where we could provide a more profound improvement, on the other hand we accepted the usage of protected regions which are considered an anti-pattern from more advanced users. In this particular case instead they helped to increase the confidence of the adopters because they had the possibility

to insert custom code as it was needed. We advocate that the MDD approach has to be tailored on the base of the maturity of practitioners.

3.3 Case Study solution

The MDD approach adopted for this project is summarized in Figure 3.2: it is based upon three different model levels. The first one, the domain model, is defined by the user by means of a textual DSL. The second one, the intermediate model, is built reworking information extracted from the first model and from binary components using reverse engineering. Finally the third model is made up of the set of Java source files and XML configuration files generated.

The user defines a model of the architecture, using the DSL syntax. In the model he specify how components in the various layers are structured. Such a model is then translated by means of an automatic model-to-model (M2M) transformation to an intermediate model. From this intermediate model another transformation occurs, this time a model-to-text (M2T) transformation, producing as output a set of textual files: Java source code and XML configuration files.

The goal is to describe the domain model through a flexible DSL, so that the language can be used to describe a broad range of situations and at the same time be particularly concise resorting to common cases when it is possible. Conversely flexibility is not a goal for the intermediate model since it is used as the basis for the generation and it is never modified manually. Considering that the DSL has to be evolved to provide a better user-experience and the M2T transformation could have to be adapted to address technological changes the intermediate model is useful to somewhat absorbe those changes without them being propagated to the other extreme (i.e., a change to the DSL could in the worst case affect the intermediate model but not the M2T transformation, while a modification to the M2T transformation would not affect the DSL). Moreover the intermediate model is designed to contain completely specified information, which simplify the M2T transformation.

The solution devised combines tested practices and techniques into a customized toolset. It represents, we believe, a typical case of MDD adoption in a small company, so it lends itself as an exemplar case study.

3.3.1 Domain model

We opted to support a textual notation for our DSL and not a graphical one due to ease of development of tools with advanced features [Völter, 2009].

One of our goals was to achieve a cost of switching as low as possible for the Java programmers already involved in the project; the use of a textual java-like syntax promised to be easier to learn than a graphical one. We considered that one of the

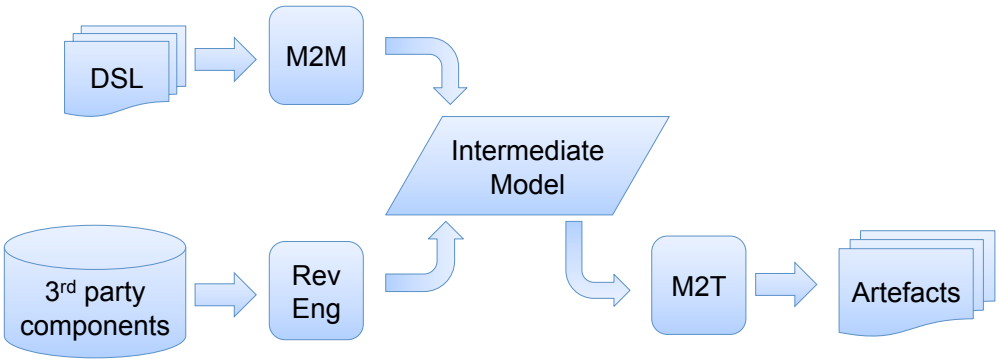

Figure 3.2. Models and transformations

main cost factors is the time needed for developers to learn the new environment and become proficient with it, consequently we decided for a notation as close as possible to the notations already used by the clients of the proposed MDD solution, who are mainly Java developers. In addition we considered concurrent engineering an important feature and while there are several tools supporting concurrent development for a textual notation (i.e., tools to execute differences analysis, automatic merge and so on), the adoption of a graphical notation would be more problematic [Leveque et al., 2009, Völter, 2009].

The DSL language we designed is used to define the structure of a stack of components adhering to the FVC architecture and it is called FVCS (for FVC-Stack). Its syntax has been defined in agreement with two driving principles: i) the whole solution is designed to be immediately familiar to its intended users, which are Java programmers, so we designed the DSL syntax to resemble Java as much as possible. ii) The language resorts on some very common cases and for this reason a set of "shortcuts" are provided. We report a small script to give the feeling of the language. In the project circa 1.0K lines of DSL code were written.

```
// import binary components
jar "path/binaryComponents.jar"
package it.trim {
   data in Abc {
      string name;
      date start;
      date end;
   }
   data out ADataOut {
      int total;
   }
```

```
   // implicitiyly refers to data in Abc
   businesscmp Abc {
      dao importedDao
      out ADataOut
   }
   service wrap Abc
   ejb MyEjb {
      Abc as "getAllXyz"
   }
   model MyModel {
      allfrom ImportedDao
      SomeOtherDao *;
   }
}
```

3.3.2 The intermediate meta-model

The role of the intermediate model as a basis for the final generation of the application model (represented by the generated artefacts) requires managing the different cases for every declaration kind and making explicit all the information provided implicitly at the domain model level through the DSL. The overall generation can be split in two transformation phases: i) a M2M transformation aiming to explicit information, ii) a M2T transformation which has to translate information to the specific target technology. By adopting the intermediate meta-model we achieved greater flexibility and simplified the M2T transformation [Völter, 2009]. The intermediate meta-model is composed by roughly one sub-meta-model for each component kind. We report in Figure 3.3 the portion of the intermediate meta-model used to represent one of the layer of FVC: Business Components.

3.3.3 Generated artefacts

From the intermediate model by means of a M2T transformation two different kinds of artefacts are generated: Java source files and Spring XML configuration files. Java source files represent the implementation of the components defined using the DSL. In general a single component may give rise to one or more Java files. For some layers of the FVC architecture a configuration file may also be required to provide information relative to the whole set of components pertaining to that layer. Depending on the nature of the generated component it can be completely generated or it can be partially generated. Source files partially generated contains all the architectural code and let the user insert the custom logic into well-defined protected regions.

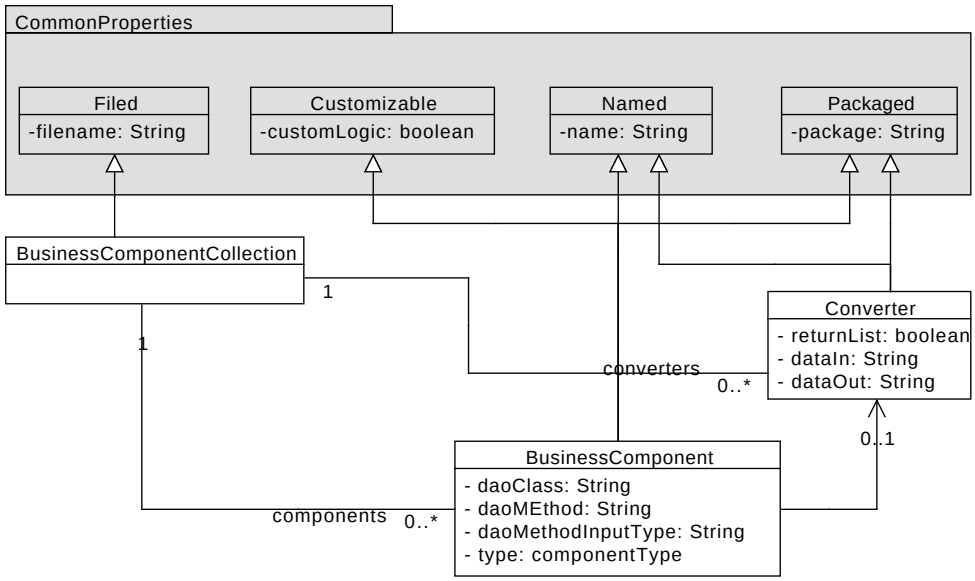

Figure 3.3. Excerpt of the intermediate Model

Configuration files define how different components are related and their scope is project-wide. There are three different configuration files, each of them contains configuration relative to a subset of the FVC layers. They are Spring XML configuration files: one is related to business components, the second one to services, the last one contains managers and controllers definitions.

Given the definition of a Business Component the application model consists of the following files:

- a Java class for the Business Component. It contains the class corresponding to the business component with the setter for the DAO and the field to hold its reference, the process method that could be completely specified (for wrapping Business Components) or just the skeleton containing a protected region in the general case;

- a Java class for the related Converter. The declared class contains methods to convert an ORM to a Data In and a Data Out to an ORM. ORMs are classes representing data in the database layer in a OO context;

- a portion of an XML configuration file for the Business Component and the related Converter providing information used by Spring to perform dependency injection.

3.3.4 Supporting tools

To implement our MDD solution we used Eclipse Modeling [Budinsky et al., 2003]. In particular we used these components of the bundle: the Eclipse modeling framework (EMF), Xtext, Xtend, Xpand.

EMF is a framework to define data models and meta-models, the structures of data models. In particular meta-models can be defined using XML, Java classes or modeling tools. The framework permits not only to define the structure of the models but also constraints that can be used to validate the model. EMF provides also a language called Ecore to define meta-models. Ecore is also a meta-model expressed in the means of itself. EMF is the technology that permits inter-operation between the different tools in the Eclipse Modeling Project. Every other tool in the project is able to operate with EMF models and meta-models.

XText is a DSL framework. It has to be fed with a DSL's syntax definition using an extended Backus-Naur Form (BNF). Starting from this syntax definition XText produces a parser, an EMF meta-model, and the skeleton of an editor. The parser is able to read a text file conforming to the DSL syntax provided and create from it a model referring to the generated meta-model. The editor will be specifically targeted at the DSL defined and will be provided as an Eclipse plug-in. Both the parser and the editor though fully working need to be customized by specifying validation rules, how to perform auto-completion, and many others refinements.

XPand is a template language able to consume EMF models to produce text files. XTend permits to create reusable definitions performing simple manipulations of EMF data. XTend is often used together with XPand to modularize some definitions that are reused many times in an XPand template.

One of the requirements was the possibility to re-use components developed with traditional techniques according to the FVC architecture but not using the designed MDD facility we developed. The first reason is that we want to able to re-use components developed by other companies that did not adopt MDD techniques. This is important because Trim (the company involved in the case study) had to cooperate with other companies working on common projects. As second point in this way we are not tied up to develop every component of every level with the MDD solution. If a specific case arises that is hard to model with the current solution and the effort necessary to enrich the solution is a lot higher than the outcome that very specific component can be developed outside with traditionally techniques and then be referred in a FVCS file, i.e., to build a component in the upper layer. To satisfy the requirement the solution is designed to be able to re-use directly compiled components consisting of both a directory containing class files or a JAR.

The module devoted to reverse engineering is able to deal with both cases (a directory of class files or a JAR). This module analyses classes looking for implementation of components of a certain FVC layer by looking at the class name, at

the interfaces implemented and at the superclass of the examined element.

3.4 Risks management

The development of the MDD solution for the FVC architecture, through a trial and error process, allowed us to better understand the key factors and the resulting benefits in developing and deploying such kind of solution in a small company with no prior experience about MDD. We first present lessons learned and later we discuss how they can be used to mitigate risks presented in Section 3.2.2.

3.4.1 Lessons learned

L1) need for intermediate level: we realized quite early in the project the necessity to introduce the intermediate meta-model. It proved to be very useful to prevent changes happening at one of the ends of the MDD solution to spread across the stages of the solution itself and affect the other end. This was important because during the development of the project we introduced many changes in the DSL syntax that were made as consequence of feedback from users. In most situations changes did not affect the intermediate meta-model but they were absorbed by the M2M transformation. Due to some changes we needed occasionally to slightly adapt the meta-model for some components but we never needed to adapt the M2T transformation due to changes in the syntax. The insulation role of the intermediate meta-model was also standing for changes coming from the technology side. As consequence both the syntax and the technology side were able to be improved and adapted separately so that we were free to let the language evolve to become more concise and at the same time we could adapt the way we produced the artifacts to meet the changing technical requirements.

L2) Convention over configuration: to adopt the principle known as "convention over configuration" lead to concise scripts. Concise scripts are good for many reasons: they are faster to write and they are faster to understand, which is also more important. They contain no redundant information so the reader can concentrate just on the particular cases, where something is not acting as usual.

L3) 3rd party integration: the ability to include components already developed in the MDD solution is critically important in industry environment. First of all it permits to not waste precedent investments and it permits a gradual transition from previous techniques to MDD. There is also another advantage: it brings confidence to be not locked in the MDD solution. If it is always possible to build components with traditional techniques and then integrate them in the models it will reduce greatly the risks involved in MDD adoption. These lessons confirm the findings presented in [Hermans et al., 2009], where several respondents wished the

opportunity of being able to import pre-existing models. It also permits a gradual adoption of the MDD approach, one of the important factors reported by [Hutchinson et al., 2011a].

L4) DSL flexibility: the MDD solution and in particular the DSL have to be flexible enough. In every technique involving modeling a certain amount of rigidity is assumed, MDD solutions needs to rely on archetypes and repeated patterns. Usually designers of MDD solutions tend to enforce too much, to limit what developers can do using the tools they create. A common pitfall is to envision in the small details how the whole solutions will be used and how the developers will have to implement the applications. This brings developers to feel unable to adapt their work tools to their way to execute the job. There is a tendency to consider developers as just another tool of the MDD solution. It proved to be more successful to think the other way around: developers are professionals and MDD designers just provide better tools that have to fit their needs and their own way to organize their work.

L5) Developer's involvement: developer's commitment is essential to obtain a successful and concrete adoption of the MDD solution. Our experience suggests that in order to obtain that developers have to be involved in the design phase considering their feedback as valuable. If developers feel that MDD adoption is going to be forced or that the solution is not flexible enough to adapt to their needs they are likely going to misuse it, causing to reduce or void the benefits provided.

3.4.2 Risk mitigation

The potential risk **R1 (tool rigidity)** has been mitigated by both L1 (need for intermediate level) and L2 (convention over configuration). The presence of an intermediate level allows adjusting the domain model and the code generation independently, thus achieving an high evolvability of the tool. Moreover the extensive use of conventions loosen the coupling among the modules of the tool, the result being a more maintainable system.

The possible **lack of developer adoption (R2)** was one of the main concerns during the development of the MDD approach. We learned that it can be mitigated in different ways: (i) relying on convention instead of configuring the utmost detail (L2) provides conciseness and relieves developers from writing repetitive patterns, (ii) the flexibility of the DSL (L4) allows the developers to easily work with the language, and (iii) the involvement of the developers (L5) in the design of the MDD solution ensured a high rate of adoption.

One of the fears of the company was switching to MDD, and then realizing it was not suitable for the project, and eventually ending up **locked in the solution (R3)**. The ability to integrate 3rd party components (L3) mitigated significantly this risk: it is alway possible to develop directly in Java any "problematic" component and integrate it into the system.

An important competitive advantage in software development consists in being able to keep the pace with technological evolution. It was not clear whether MDD could jeopardize this ability of the company; the risk is to increase the inertia toward the evolution of the application to **adopt new technologies (R4)**. The existence of an intermediate level (L1) let the generation phase to evolve easily. Moreover the extensive use of conventions (L2) avoids over-specified models by introducing technology-dependent information, so mitigating the risk of evolutionary inertia.

3.5 Acceptance assessment

The reception of the MDD approach was evaluated by means of a questionnaire which was addressed to the different roles involved in the project who are employed at Trim. The goal was not to evaluate our solution but how participants in different roles reacted to MDD. The questionnaire was carefully developed by the two academical authors of this paper.

This section presents first the questionnaire, later the answers divided by topic are analysed.

3.5.1 Questionnaire definition

The survey was addressed to the Trim's personnel involved in the project. The MDD solution was developed at the Politecnico di Torino with the constant feedback of the company. Later the tool was used and maintained at Trim. In particular there were five Trim's workers involved and all of them accepted to fill in a survey tailored for the specific role they covered in the project. One of the workers is the manager (who is also a co-author of this paper) who contributed to develop the initial design of the MDD solution. A second one is a software architect with a broad and deep technical culture: he contributed with technical comments to the initial design and he was appointed with the duty to maintain and evolve the tools. The third worker is the project manager of the projects which used the tools since the first pilot and later in production. Finally there are two developers involved in those projects. The objectives of the questionnaire are to evaluate the results and in particular to ascertain possible deficiencies of the proposed approach, thus providing insights about the reception from the different point of views.

We adopted the participant-observer case study [Yin, 2002] because we needed the feedback from the personnel involved. Because we are not evaluating our approach to demonstrate its efficacy we consider the potential bias caused by the interest in the project to be not relevant.

Questions are divided into three groups which focus on different aspects of the project. The first one is about results evaluation (questions R1 through R11), the

second one is about acceptance (questions A1 to A5) and finally the third one is about development process changes (questions P1 to P6). Overall we formulated 22 questions that are reported in the leftmost column of table 3.1. While the questionnaire was originally written in Italian, we present here the corresponding English translation. The questionnaire included both closed (C) and open (O) ended questions; the second column of the table describes the type of question. The close questions have been formulated in the form of assertions with which the respondent had to express his agreement according to a five point Likert scale [Oppenheim, 1992]: *Strongly disagree* (1), *Disagree* (2), *Neither agree nor disagree* (3), *Agree* (4), and *Strongly agree* (5). The typical encoding, used also in our analysis, is the integer number reported in parenthesis.

3.5.2 Discussion of responses

Table 3.1 reports, for each question, the answers provided by the different roles. Since the set of questions administered varied according to the role, there are combinations of role and question that have no response, such lack of answer is represented by void cells . For close-ended questions the cell contains the level of agreement encoded with an integer from 1 to 5 as shown above. The check mark, for open-ended questions, indicates that the question was administered to that role, who provided an answer (the full text of the responses is available in Appendix 3.8). When the respondent did not provide any answer we report "NA". In particular developers used the possibility to not give an answer in two different cases because they were less involved into the project with respect to other roles and the changes due to the adoption of the MDD solution affected them marginally as we will report while analyzing answers on process changes. They were less involved because they were working mainly on the presentation layer, which is the layer excluded by the MDD solution. Anyway their role requires to use components generated by our solution and by means of specific questions we wanted to verify that its adoption is almost transparent to them.

3.5.3 Results evaluation

Typically when introducing modeling solutions there is the concern to cause rigidity to the development process: we rest reassured that the tool was in no-way a limit during the development (R1). The tool managed to reduce repetitive work (R2) although not to completely eliminate it. Anyway the reduction was sufficient to shorten significantly the development times (R3). The solution was considered also a tool useful to design globally the system and maintain an overview of the whole application (R5). As far as defect reduction is concerned (R4) the answers indicate that defects were not reduced by the adoption of the solution. First we

Gr.		Question	O/C	Role			
				M	SA	PM	D
Results	R1	The tool constituted a limit in any way during the development	C			1	1,1
	R2	Tool usage reduced repetitive work	C			4	4,4
	R3	Tool usage reduced development time	C	4	4	4	NA,NA
	R4	Tool usage reduced the number of defects in the developed applications	C	NA	3	3	NA,NA
	R5	The FvcGen approach supports the design phase and provides an overview of the system	C	5		4	
	R6	The approach improved maintainability	C		3	3	
	R7	The tool is easy to use	C			4	
	R8	The tool requires a quick learning phase	C			4	
	R9	The Fvcs DLS syntax is easy to learn	C			4	
	R10	Which aspects would you consider as critical to realize a similar project?	O		✓		
	R11	How could the solution be improved?	O			✓	
Acceptance	A1	Using the tool is professionally stimulating	C		4	4	4,4
	A2	Which were the major resistances to the adoption?	O	✓			
	A3	What could be done to favor the tool's acceptance?	O	✓			
	A4	Which are the most critical aspects for the adoption of a MDD solution in an organization?	O	✓			
	A5	How do you evaluate the adoption for the solution's users?	O		✓		
Process	P1	The adoption of FvcGen changed the way you work	C				1,1
	P2	The transition to the new development process has been quick	C	4			
	P3	The transition to the new development process has been very easy	C	3			
	P4	Which are the critical aspects for the adoption of FvcGen?	O				✓
	P5	Was the development project changed by the introduction of FvcGen? How?	O	✓			
	P6	Introducing FvcGen do the competences required for developers change?	O	✓			

Table 3.1. Answers to acceptance assessment questionnaire for different roles: M - Manager, SA - Software architect, PM - Project Manager, D - Developer. O/C = Open/Closed question. The numbers reported refer to a Likert scale ranging from 1 (Strongly disagree) to 5 (Strongly agree).

note that respondents were mainly concerned with post-release defects, and we interpret the result as an indication that architectural errors were not reduced. We believe that low-level errors (typically those caused by misconfiguration or mistypes) quite common in hand-written code are eliminated by our solution. Such defects are show-stoppers and are routinely fixed before release causing a slow-down in the development and annoyance for developers. Though the company does not collect measures at this level and therefore no evidence is available to support this hypothesis. Apparently there is not a particular advantage in terms of maintainability (R6). This outcome can be explained by observing that the project has not entered the

maintenance phase yet; therefore the answer to the specific question was a conservative one. There is a set of questions (R7, R8, R9) specifically addressed to the principal user of the tools (the Project Manager) and in particular of the FVCS language. The answers confirm that the tools and the language were practically usable and quite easy to learn. These characteristics were of course a key factor for the choice of Trim to adopt the solution. Finally analysing answers to the last questions we can get more general advice. The Software Architect considered the approach used (R10) to be correct and repeatable for similar applications, although he expresses doubts about the applicability of a similar approach to architectures that do not exhibit such a rigid multi-tier structure. The Project Manager instead points out that the solution could be improved (R11) by making it more easily deployable and reducing dependencies issues. As a side note he indicated, also during the development of the project, the necessity to increase flexibility e.g. by making the destination of generated artefacts for different components and the definition of Java package names directly configurable.

3.5.4 Acceptance

Because we consider the solution acceptance from all roles working at the company an absolute key-factor for the success of the project we realized a set of questions specific for this topic. We first asked the technical figures involved to find out if the considered this experience as professionally interesting (A1). We obtained positive answers by all roles. Then we asked the Manager and the Software Architect for their considerations. The Manager considers an hindrance to adoption (A2) the fact the developers tend to dislike technical solution proposed by external subjects. In this case they accepted it because they could see concrete benefits early. As a second point some issues in the tools deployment to different Eclipse installations were considered annoying. He suggested in order to favor the adoption (A3) to organize lessons on the use of the solution. Moreover we asked for advices on the adoption not specifically tailored to this experience. The Manager affirmed that the MDD solution should be to reused across other projects (A4). He specify that could be not always possible because, while the customer of this project would appreciate the adoption of a similar solution, other customers could ask the company to use a specific development approach not based on the MDD solution. The Software Architect considered the reaction of the solution users (A5) positive but he suggested that it could be subject to personal tastes; he elaborated further that while the people involved in the project appreciated it, it could be the case that developers prefer to develop manually most of the code in order to maintain strict control.

3.5.5 Process changes

We also wanted to evaluate how the development process was affected by the introduction of MDD in a small company for its first time and which were the changes. We found that transition was quite smooth: quick (P2) and with no particular difficulty (P3). As far as developers are concerned, since their work is essentially centered in the presentation layer, the adoption of the MDD approach is almost transparent to them (P4) and therefore their job is not affected (P1). A major difference emerged for the Project Manager: he is now able to do on his own a greater amount of work; in particular he can develop, by means of the tools, the whole back-end of the applications (P5). This is partially justified by the fact that he is "shielded" by many technical details related to the specific technologies involved and as result he is less concentrated on technological aspects and can spend more attention on the business logic. Finally, prerequisite on technical competences for the developers involved in the project can be relaxed (P6). This represents an important benefit for the company that can hire workers that are not necessarily expert in every single technology of the architectural stack.

3.6 Related work

Insights on the diffusion of MDE are reported by a large study from Hutchinson et al. [Hutchinson et al., 2011b] conducted by means of questionnaires and interviews. Another study about MDE diffusion (limited to Italian companies) was presented in the previous Chapter. Those studies are interesting to understand the phenomenon at large but do not reach the level of detail that case studies and experience reports permit to achieve.

While the literature includes several experience reports on MDD adoption, most if not all of them concern studies in the context of large industrial setups and companies with a medium to long experience in the field.

A notable example is Baker et al. [Baker et al., 2005] who report on the competencies developed at Motorola after 15 years since the adoption of MDE. In their experience the major obstacles in adopting MDE stem to the lack of a well-defined process, lack of necessary skills and inflexibility in changing the existing culture. Another account from a 5-years project at the same company can be found in [Foustok, 2007].

Fleurey et al. [Fleurey et al., 2007] report on the 10-years' experience with MDE developed at Sodifrance; the focus is on migration projects, where the benefits w.r.t. conventional techniques can be observed after an initial period, e.g. the first code could be delivered only after 10 months from project's beginning. In addition they present a cost-benefit analysis and suggest the presence of profitability threshold in

terms of project size.

Hen-Tov et al. [Hen-Tov et al., 2009] describe a project with enterprise software; while the software category is similar to our case study, their approach requires an initial development effort of 10 man-years.

Hutchinson et al. [Hutchinson et al., 2011a] report lesson learned from adoption of MDE in three large multinational companies (a printer company, a vehicle manufacturer and a manufacturer of electronic systems). They conclude that important enablers for the succes of MDE in those large companies were: adopting a progressive and iterative approach, obtaining organizational commitment, motivating users, organization flexibility from the whole company and the presence of a business focus motivating the adoption of MDE.

The above reports concern MDD solutions that require a very long setup time; one article from MacDonald [MacDonald et al., 2005], which analyzes development through MDD of a component for a legacy systems, describes an approach requiring a low initial development effort. We think that small-companies have to choose a different way to MDD adoption than large ones and it seems to be neglected by the current research trend. For this reason we try to adopt some research approaches used in large-companies and adapt them to a case study undertook in a small one.

There are other works that studied MDD through a survey as we did. For instance, a survey on the success factors of DSLs adoption conducted on a large set of projects spread among many different companies [Hermans et al., 2009]. The scale factor difference between our work and that one was a crucial aspect affecting the way we designed our questionnaire and how we analyzed the results. Due to reduced number of people involved in the project we could not perform statistical analysis. Moreover our work differentiate the people involved in the survey by their roles while [Hermans et al., 2009] does not. Another work is from Staron [Staron, 2006] who proposed a questionnaire to personnel of two companies considering MDD for adoption, one having already undertook a pilot, the other not yet. It emerges that the three most important factors influencing the decision for adoption are i) availability of modeling tools, ii) cost of introducing the modeling technique to the process, iii) cost of creating models during software development. Once again companies considered are really large organizations. Shirtz et al. [Shirtz et al., 2007] reports considerations about successful way of convincing management to adopt MDD, their considerations on this topic are not the central part of the paper and are anyway related just to large companies.

Finally a review of experiences on MDE applications from Mohagheghi and Dehlen [Mohagheghi and Dehlen, 2010] contains considerations on effects on code quality (but not supported by data) and productivity while we are trying to perform an analysis of the effects on the development process and reactions by different roles involved.

After we performed this research a new article on application of MDD in small

companies was published [Cuadrado et al., 2013]. The authors agree with us on the lack of research of MDD deployment in small companies. The motivation is that, while the problems encountered from small companies are typically the same of the large ones, the way to face them are rather different because different resources and different priorities characterize the companies. In their work Cuadrado et. al used two case-studies performed at two different companies. While results were positive, the companies did not fully embraced MDD as consequence. In particular small companies have to consider with attention the cost of developing MDD solutions and decide if their worthy for the task at hand.

3.7 Summary

In this chapter we discussed the adoption of a MDD solution in a small-company with no prior experience with such techniques. We described the specific context and reported and motivated the principles applied to design the solution. In particular we stressed the importance to deliver high-quality supporting tools in order to guarantee an acceptable productivity. In a small company it is particularly important to achieve early benefits as result of small initial investments. In order to attain such goal we strived to (i) build an environment as familiar as possible for the prospective users and (ii) avoid any rigidity that could hinder the reactivity to requirements or process changes, which is one of the competitive advantages of small companies in respect to larger ones.

We investigated the perception of the solution by the personnel involved in the project. We could confirm the good results obtained and we gained considerable insight about deficiencies and ways to improve our approach. The respondents emphasized two main key success factors: very high-quality tools and flexibility. While these aspects were considered satisfying in this case study, they are so important to need still more attention.

Given the cross-sectional nature of our study, we did not consider long term effects. In particular no evidence could be found regarding the impact of the proposed MDD solution on maintainability and defect reduction: we plan to investigate more these aspects in future research.

As future work, we plan to evaluate again MDD approaches in small companies in the context of different kinds of applications investigating more in depth process changes.

Item Role	Question Response
R10	*Which aspects would you consider as critical to realize a similar project?*
SA	The current version is suitable for the development of multi-tier applications, therefore for such kind of projects FvcGen does not bring any criticality. On the other hand, if the architecture were less layered then the use of the instrument *as-is* could introduce some issues. A further re-engineering of the tool could be required.
R11	*How could the solution be improved?*
PM	Right now it appears to be too constrained both in terms of generated code and used libraries. It was very hard to make it compatible with current work instruments (e.g. different versions of Eclipse and Ant)
A2	*Which were the major resistances to the adoption?*
M	In general developers are not likely to accept technical decisions taken by others. In this case, though, who used the tool immediately gained real benefits. The problem, which could limit its adoption in future projects, is the complexity of the installation procedure, due also to the several version of Eclipse.
A3	*What could be done to favor the tools acceptance?*
M	No idea about the tools. Though we could organize training sessions.
A4	*Which are the most critical aspects for the adoption of a MDD solution in an organization?*
M	For sure process and technologies ought to be standardized. For a company like ours, this is very difficult since customers often impose their techniques. In a context such as our current banking customer I see the adoption of this technique as very feasible without particular criticalities (but those related to natural resistance towards change).
A5	*How do you evaluate the adoption for the solutions users?*
SA	Positively, although it is heavily dependent on the person that use it (some are more productive when they work according their habits, others are more open to innovation)
P4	*Which are the critical aspects for the adoption of FvcGen?*
D	I did not use it directly (*both developers replied in this way*).
P5	*Was the development project changed by the introduction of FvcGen? How?*
M	Changing the process was exactly the goal of this collaboration. In particular we wanted the team to be less focused on the technological aspects and closer to the business issues. The solution actually collapsed on a single person the development of the generated components.
P6	*Introducing FvcGen do the competences required for developers change?*
M	Yes. All the generated portion hides precisely the technological complexity.

Table 3.2. Responses to open ended items (translated from Italian into English)

3.8 Appendix - Responses to open ended items

Acknowledgments

We want to thank the personnel at Trim who took time to answer the questionnaire. We are also grateful to the anonymous reviewers for their comments and suggestions which helped us to improve this study.

Chapter 4

Modeling adoption in large company: the CSI case-study

In this Chapter we investigate first time adoption of MDD in a large company and its sourrounding ecosystem. It was a joint work realized with Marco Torchiano, Maurizio Morisio, Mauro Antonaci, and Paolo Arvati.

Results of the work presented in this Chapter contributes to answer the thesis research question **RQ A.5**.

4.1 Introduction

A software ecosystem can be defined as: "a set of actors functioning as a unit and interacting with a shared market for software and services together with the relationships among them" [Jansen et al., 2009]. We believe that the term market should be taken in its most general sense: the set of relationship existing between (type of) actors representing any form of exchange. Typically, in a software ecosystem, actors exchange software artefacts, services, and of course money. As for a natural ecosystem, also in software ecosystems variations in the behavior of one actor (species) cause reactions from other actors and alterations of the overall environment. In such a kind of complex systems both changes originating from the participants or perturbations coming from external factors (e.g., the economic conjuncture) trigger chain reactions by many of the actors which take part in the ecosystem; the result is either the creation of new relations, modification of the existing ones, or destruction of some of them. Therefore the shape and behavior of an ecosystem as a whole is extremely difficult to predict and govern.

4.1.1 Context

This work focuses on an ecosystem centered around a large publicly owned organization, CSI-Piemonte (Consortium for Information Systems), considering the relations between departments, with ten of sub-contractors and with the customers (hundreds). CSI-Piemonte (CSI hereinafter) was founded in 1977 with the aim of promoting the modernization of local administrations by using IT-based tools to create information services and systems. It focuses on the development and operation of Information & Communication Technology projects for the Piedmont's Public Administrations (PAs), providing services for citizens and businesses.

CSI is a consortium with over 100 members, most of which are PAs: the Piedmont Region, several Provinces, and many municipalities. Other members are universities, hospitals, and local health agencies. Many of the members of CSI are also among its customers. Services are developed for many of the PAs of Piedmont which counts over 4.5 millions of inhabitants distributed across over 1200 municipalities (this high number is due to the orography of the region).

The goal of this work is to document the evolution of the CSI-centered ecosystem over a span of 5+ years, during the introduction of a new development technology: Model-Driven Development (MDD) [Mellor et al., 2003]. The introduction of MDD started in 2008 and is still in progress. The new technology induced several changes in the ecosystem, concerning both the role of the actors and their interactions. In parallel and tightly interlocked with the evolution of the ecosystem we will follow the evolution of the MDD supporting toolset.

4.1.2 Motivation

The presence of a central catalyst and a shared technology, bringing specific benefits to the different participants are fundamental to create a cohesive ecosystem, motivating everyone to favor the success of the technology and, as consequence, benefiting the whole ecosystem.

While the entity at the center of the ecosystem – i.e. CSI – was able to build this cohesive ecosystem and earn the support of the participants, still, a huge effort had to be spent to steer the ecosystem and operate a mindset change, winning the inertial resistances. CSI had to initially spend a huge effort not only developing the tools but also investing in complementary aspects (IDE integration, documentation, support, lobbying). However the success in the transition was made sustainable in the long run by the progressive involvement of other actors that helped in a increasingly more active way as progressing in the our story.

Creating the fertile pre-conditions and the determination of the steering organization are however not enough for the survival of the ecosystem. They are complex systems where entities with possibly conflicting goals and a number of inter-relations

co-exist. In this kind of environment technology could play the role of the enabler for a mindset change but many other aspects are crucial and have to be properly considered: among them we wish to underline necessary competencies, organizational aspects, economic aspects. In such complex systems, where so different aspects have to be considered, it is hard to forecast the effects of changes and the long-term results of actions. This could lead to unanticipated benefits (like the spreading of MDD competencies in a local area) but to problems as well. We therefore think that steering actors or simple participants in similar ecosystems could benefit from a few guidelines, emerging from successful cases of paradigms transitions operated in software ecosystems.

4.1.3 Organization of the work

The chapter is organized as follows: section 4.2 introduces the method adopted to conduct the study, section 4.3 describes the evolution from an historical perspective, section 4.4 analyzes and distills different motifs that characterize the successful transformation of the ecosystem. Later we discuss the main aspects of the transformation of the ecosystem (section 4.5), then we present the related works (section 4.6) and finally we draw our conclusions (section 4.7).

4.2 Method

The collection of information was carried on over a period of two years starting in April 2011. At that time a collaboration between the Software Engineering group at Politecnico di Torino and CSI Piemonte started, which focused on the development of a model versioning infrastructure to be used for the MDD tool suite.

During the collaboration the researchers became aware of the articulate history related to the conception, development, introduction, and deployment of the MDD solution and decided to undertake an additional hermeneutical research effort focused on the evolution of the ecosystem centered around the MDD tool suite. The team – the authors of the present work – is composed of a group of academic researchers and a group of industrial members.

The research method adopted in this work is essentially of interpretive nature [Klein and Myers, 1999]. In particular the investigation is based on a single case study that lasted almost six years, which encompasses several hundreds individual software development projects.

The collection of materials occurred in several different occasions.

- An initial series of meetings approximately taking place with bi-weekly frequency, they where originally intended to understand the architecture of MDD tools for the purpose of collecting the requirements for the model versioning

infrastructure. Those meetings provided an initial overview of the ecosystem and its historical evolution.

- A workshop was organized in July 2011 for the announcement of the release of MDD-tools as open source software, the researchers participated in this workshop where accounts of experience with MDD tools by third party developers were presented. The feedback from external subcontractors allowed us to confirm the information collected from within CSI.

- Two meetings were organized to focus on the historic-technical perspective of the ecosystem, the meeting were conducted by the researchers in the form of unstructured interviews where the industrial participants were asked in general about the ecosystem evolution and specifically about the socio-technical aspects that characterized it.

- Eventually a working document was produced to summarize what emerged during the focused meetings and served as the reference for discussion and clarification, which took place mainly via email and telephone.

- After the first version of the work was available, the researchers conducted two semi-structured interviews with developers who actually used the MDD tools in order to confirm or refute the interpretation.

The goal of our investigation is to describe a complex and large local ecosystem and document the main patterns that emerge during its historical evolution. Though our approach is similar to a case study [Runeson and Höst, 2009], the breadth, temporal duration, geographical extension, and moreover the interpretive nature make it depart from the usual understanding of that kind of study.

The type of data collected in the previous events range from informal notes taken on paper, to detailed notes taken e.g. with text editors, to structured notes taken e.g. with mind-mapping tools, to interview transcripts. Due to the heterogeneity of the materials and the mainly interpretive nature of our work we decided not to use common qualitative methods (e.g. coding techniques)[Seaman, 1999], which are more suitable for hypothesis confirmation – i.e. a positivist approach – and homogeneous materials.

We summarize the main features of our work with reference to the basic principles of interpretive field research proposed by Klein and Myers [Klein and Myers, 1999]:

Fundamental Principle of the Hermeneutic Circle: it is assumed that "movement of understanding is constantly from the whole to the part and back to the whole" [Gadamer, 1976]. Our understanding started with specific technological aspects concerning model lifecycle and extended to the

whole ecosystem, the several iterations allowed us to achieve a satisfying comprehension.

Principle of Contextualization: in reporting the evolution of the ecosystem we strive to provide as much contextual information as possible to show how the observed phenomena emerged.

Principle of Interaction Between the Researchers and the Subjects: several concepts presented in the chapter emerged through the interaction among the researchers and the industrial authors. The very idea of considering the MDD tools, its support team, the development teams using it as an ecosystem emerged during the meetings and was not originally present in the data.

Principle of Abstraction and Generalization: the first part of this chapter (see section 4.3) reports and idiographic interpretation of the CSI ecosystem evolution, while the following part (see section 4.4) attempts to abstract a few general patterns that may attain a wider, nomothetic validity. We do not claim any statistical representativeness of our generalization, its validity relies "on the plausibility and cogency of the logical reasoning used in describing the results from the cases" [Walsham, 1993] and in the substantial agreement with findings from other works available in the literature.

Principle of Dialogical Reasoning: given our role of authors, we may declare the naïve expectation of the adoption of MDD tools to spread though the ecosystem solely due to its pure technical excellence. We actually confronted such preconception in two ways: when data actually supported it we looked for some additional objective measure, on the contrary we reported other factor affecting the diffusion of MDD as they emerged from the analysis of data.

Principle of Multiple Interpretations: the main source of information for this investigation relies on the two industrial coauthors who both belong to the MDD support group. Additional viewpoints come from the third party developers presentations during the workshop and from two interviews with individual developers belonging to the group of MDD tools users.

Principle of Suspicion: the researchers' group often discussed about the information provided by the industrial side and always concluded that the trust relationship was well deserved.

The assessment of the validity of the results is an important methodological part in any experimental and more in general positivist research. We intend to stress that our effort is essentially of interpretive nature, therefore the generalization

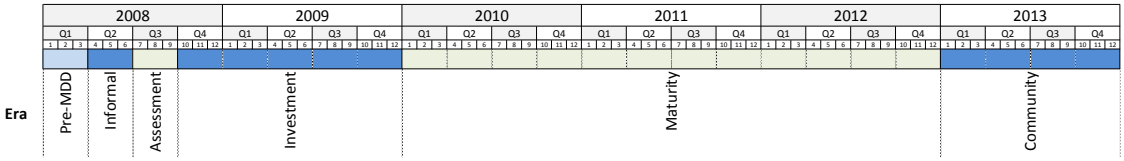

Figure 4.1. Overall timeline

effort cannot be evaluate in terms of external validity. The motifs reported in section represent an attempt to abstract the key features of the ecosystem evolution, though it is not possible to claim their general applicability to other contexts. They can be compared to existing results in the literature and to the authors' experience, and it is possible to argument – though not prove – their potential validity in similar settings.

4.3 History

The software ecosystem revolving around CSI started its evolution, due to the introduction of the MDD approach, back in 2008; we document its evolution over a five years period until 2013, time of this writing. We start by presenting the environment before the introduction of MDD and then document the evolution through five main eras. Figure 4.1 shows the overall chronological layout of the eras:

- **Informal:** initial MDD tools are developed and used in an ad-hoc manner;
- **Assessment:** commitment to evaluate a possible platform for enterprise-wide adoption;
- **Investment:** development of MDD-Tools platform and diffusion within enterprise boundaries;
- **Maturity:** involvement of contractors and enhanced support;
- **Community:** the community take responsibility for the MDD-Tools.

In order to represent the ecosystem throughout its evolution we use a set of diagrams – one per era – that represent the members of the ecosystem as ovals. The artifacts and services exchanged by the members are reported as arrows from the producer to the consumer. We depict sub-systems – i.e. groups of members that play a specific role – using dashed ovals. Sub-systems can be constituted by single working groups (e.g., the software engineering group), by categories of personnel (e.g., business analysts) or categories of organizations (e.g., sub-contractors). For

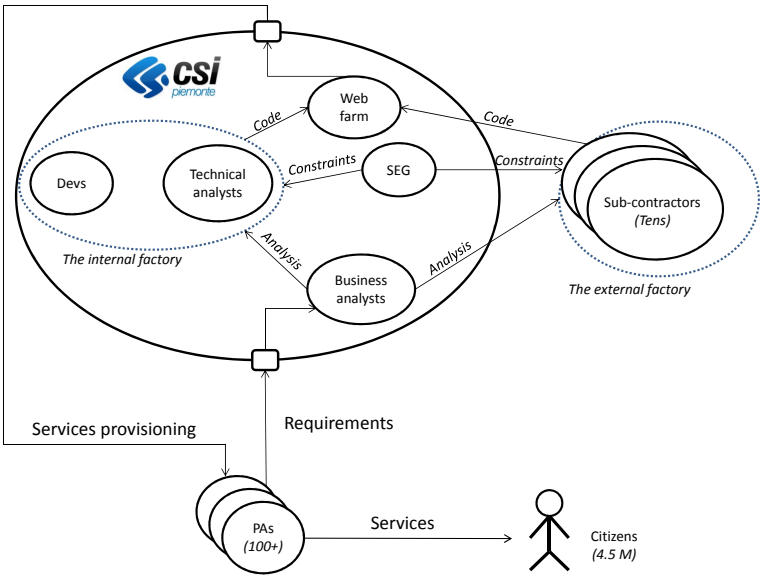

Figure 4.2. Ecosystem before MDD

every era we highlight with thick (red) lines the changed or new elements with respect to the previous era.

During the different eras it is possible to observe how the new paradigm starts as an autonomous initiative of a small group of developers (*Informal*), then it gains the attention of the CSI management (*Assessment*), it is adopted within the boundaries of the CSI organization (*Investment*). Later the change affects the whole ecosystem (*Maturity*) and finally the responsibility for the solution is moved away from the central organization to be distributed across the ecosystem (*Community*).

4.3.1 Before MDD Era

Period: *until March 2008*

The initial configuration of the ecosystem can be observed in Figure 4.2.

In CSI different kinds of applications are developed. Among them a large number are web applications to be plugged into web portals for several different public administrations. These web applications often need to comply with rigid internal rules, which include coding standards, building standards, graphical standards, and rules for other technical aspects such as the mechanisms for authentication. The standards are in place to make the development process more robust and reliable, obtaining predictable costs, and ensuring the products meet given quality standards.

Development rules are mainly defined by the Software Engineering Group (SEG).

The presence of rules and conventions is also motivated by the fact that CSI does not offers software development services only, but provides hosting to some of its larger institutional customers too. Thus the company aims to produce applications based on small number of technological infrastructures, with the goal of reducing the effort for deployment and maintenance of the web farm. The web farm is an infrastructure managed by CSI and used to host a large number of customers' web applications.

The organization strives to have a software factory where the kind of applications typically developed (web applications) can be produced according to a precise and repeatable process. The software factory is partially internal (consisting of technical analysts and developers employed directly by CSI) and partially external (i.e. subcontractors).

Customers describe requirements to business analysts from CSI; who produce an analysis document that is delivered to the *software factory*. The factory builds the application, according to the analysis and adhering to the development rules. Once produced, the application is typically passed to the system admins who are responsible for the deployment in the internal web farm.

The reported baseline productivity – at this stage of the ecosystem evolution – was 15 function points per person-month, considering the overall project. While, focusing on the bare development activities (within the software factory), the productivity was 30 function points per person-month. For the computation of functional size CSI uses IFPUG function points [Ifpug, 2012].

4.3.2 Informal Era

Period: *between April 2008 and June 2008.*

The SEG developed a tool to generate the skeleton of services. The ultimate goal was to simplify the startup process of projects developing new services and to reduce both the effort and the errors – frequently due to a *copy & paste* approach.

The tool was named *csiskelgen* and it was initially intended to be a simple template-based generator. During the development the SEG realized that they could produce a more general and therefore useful tool with a small additional effort. For this reason they decided to make it more flexible adopting a Domain Specific Language (DSL)[1] for the description of the services to be generated.

[1] A Domain Specific Language is a language with a limited expressiveness designed to express concisively a certain aspect. Famous DSLs include HTML, CSS and Latex. Recent tools highly reduced the cost of developing this languages, augmenting significantly the number of practicioners realizing them. For more details see [Fowler and Parsons, 2011]

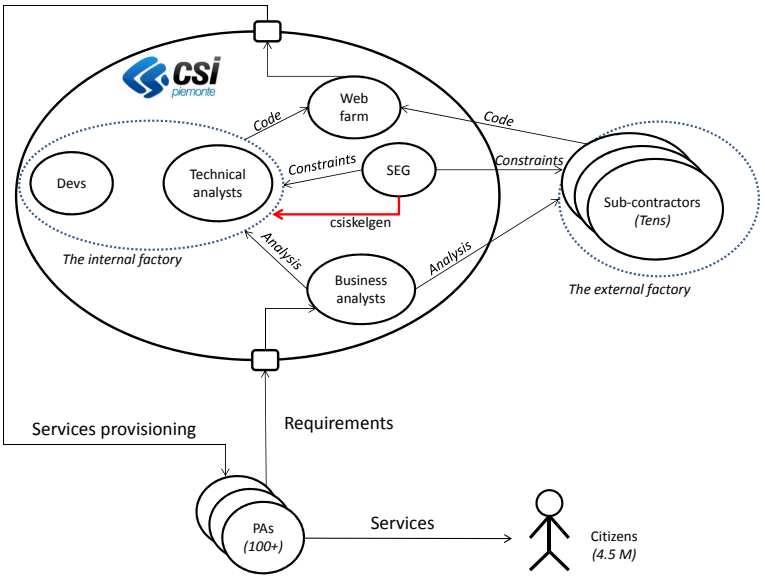

Figure 4.3. Ecosystem during the Informal era

The following two usage modalities were supported:

basic modality generate the skeleton of the service and from there develop the application without further using the tool,

advanced modality define through a model the interface of the service and some aspects like security and transactionality. From this model code could be obtained and application-specific logic could be written within protected regions. Then the model could be later modified and the code re-generated with the protected regions preserved. So the tool could be used along all the duration of the project. Users do not just generate an initial skeleton but they build a model of the service then they progressively refine it during the whole life of the project.

At this stage the company management was largely unaware of the sperimentation with MDD techniques: this was regarded as a technical implementation detail, known only to developers using it. This was possible because project management let the developers pick their own tools and this stage those tools were not regarded as an important asset or a mean to execute a organizational transition. So the developers working in the company could freely decide whether to use the tools. A large portion of them decide to adopt the "advanced modality".

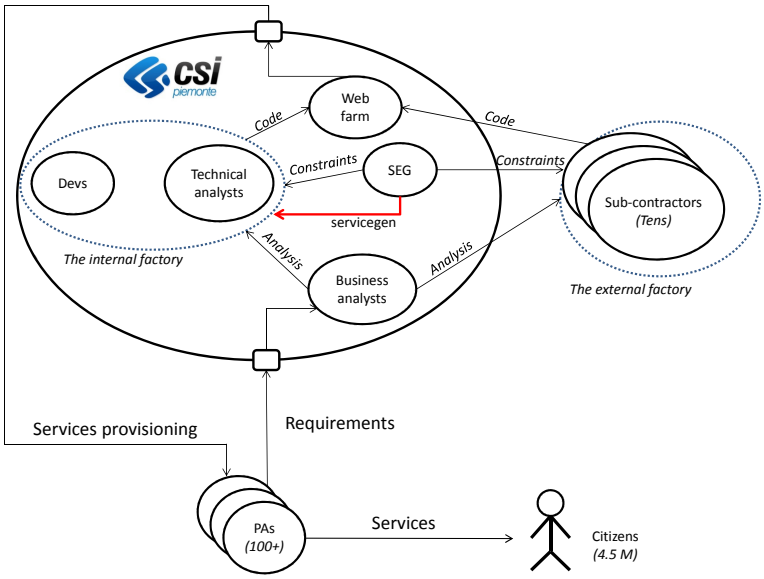

Figure 4.4. Ecosystem during the Assessment era

This first era of the ecosystem is described in figure 4.3. The main novelty is represented by the *csiskelgen* that is provided by the SEG to the developers in the software factory. An important *motif* in this era is represented by the adoption of the MDD approach, conducted on a voluntary base (see Sect. 4.4.1).

No precise data is available about productivity but the developers reported the impression of benefiting from adopting MDD. Actually the diffusion at this stage was primarily driven by an immediate productivity gain perception.

Still some developers did not like to give up the full control on the code but they felt the trade-off was worth while because in return they were shielded from some technical details about more difficult and less creative aspects, e.g. securitization.

4.3.3 Assessment Era

Period: *between July 2008 and October 2008.*

In the previous era a basic MDD solution was started autonomously by developers and began to spread exclusively because of word-of-mouth. At this point the CSI management become aware of this solution and decide to conduct an evaluation of MDD technologies. The goal of the evaluation is to drive possible new investments on a more complete toolset.

In this timeframe the SEG evaluated different software solutions for the rapid

development of business web applications. They reached the conclusions that bending those tools to obtain applications as expected by the CSI infrastructure (CSI does not only develop applications abut also host them in its own web farm) would have been very difficult. The company realized the only feasible solution was to craft its own tools. It was clear upfront that the investment to be done would have been relevant.

The SEG decided to initially focus on the evolution of *csiskelgen* tool before facing the more challenging task of building a full-fledged web application generation tool chain. The evolution of *csiskelgen* was named *servicegen*. The major enhancements introduced were:

- the model editor was improved: while before the reflection-based editor for EMF[2] models were used a specific one was now developed;
- only the "advanced mode" – supporting a completed MDD round-trip process[3] – was kept while the "one-shot" approach was discontinued;
- the tool was integrated into Eclipse, thus providing a complete platform and uniform user experience;
- service orchestration modeling was added, covering an aspect traditionally considered hard by developers.

Figure 4.4 shows the ecosystem during this era. At this level the only noticeable difference is the shift from *csiskelgen* to *servicegen* but another crucial aspect is that the support for the initiative is growing inside the company boundaries and spread from developers to the management. Moreover something important happened under the hood: enough confidence was gained about the maturity of the enabling technologies, the decision was made in favor of toolsmithing, and the technological platform – Eclipse EMF – was selected. In particular the decision of developing their own tools (see Sect. 4.4.2) represented a fundamental decision for the future evolution of the ecosystem.

[2]EMF stands for Eclipse Modeling Framework (see `http://www.eclipse.org/modeling/emf/`. It includes a complete suite of interoperable components to build personalized MDD solutions

[3]By the term *MDD round-trip process* we indicate the typical development process which starts by defining models, then generate code from them, permit the manual customization of special region of the generated code and going back to modify again the models, re-generating new code from them without losing the manual customization of the code. This is a circular, iterative process which starts from models and permit to later operate again on models. It is the alternative to the one-shot generation approach in which code is generated from models once and later always edited directly, without further operating on the models.

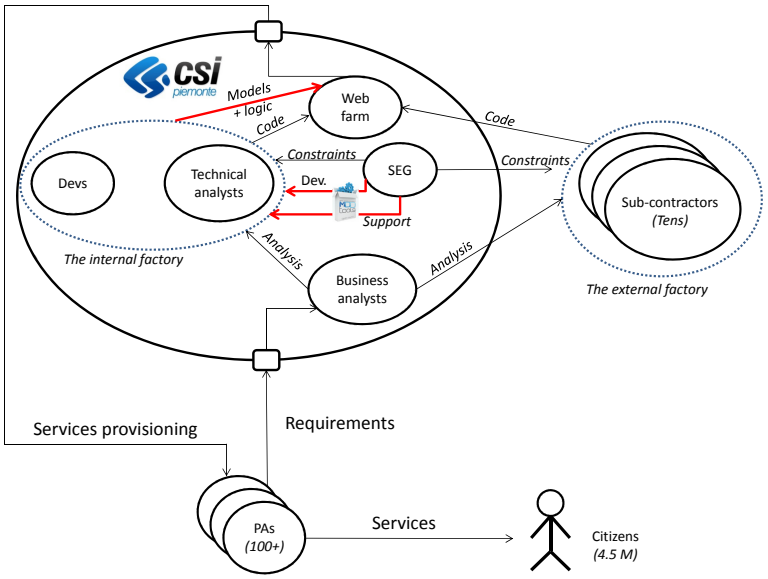

Figure 4.5. Ecosystem during the Investment era

The solution emerging was based on the definitions of a set of meta-models using the Ecore metamodel. Ecore is the solution typically used to describe meta-models when using the EMF platform. It is also the reference implementation of the OMG's EMOF (Essential Meta-Object Facility). The EMF models describing the single applications (and adhering to the designed meta-models) where developed through specific editors created in these phase. From models source code was generated using Xpand[4], a templating system interoperable with EMF models.

In terms of adoption of the MDD approach became more popular, but still limited to services and in particular to orchestrated services.

4.3.4 Investment Era

Period: *between October 2008 and December 2009.*

Given the technical viability of the MDD approach has been verified in the previous era, the company management launched an internal project for the development of a suite of model-driven tools (named *MDD-Tools*). The goal of this project was both to improve and consolidate *servicegen* but also to build the tools required to

[4] `http://www.eclipse.org/modeling/m2t/?project=xpand`

develop web applications presentation layer (*guigen*) and data access layer (*datagen*).

A significant investment in the MDD-Tools project was contributed as part of a project (α). After project α a set of pilot projects were conducted supporting with concrete evidence further decisions; the essential productivity features are presented in Table 4.1. The company measures regularly the number of function points of each process using an approach based on IFPUG.

All these projects were completed in time and reached their goals. The projects had good productivity (compared to the usual productivity measured at the same company), even better than the expectations; this is was particularly welcome because many of the developers involved had not prior experience with MDD. Circa 20 developers learnt how to use the MDD tools, and at the end of 2009 some teams were able to utilize MDD Tools without the direct involvement of the authors of the tools.

A development group (within the SEG) was created and put in charge of developing the *MDD-Tools* project. As a consequence, a rigorous development process for the tools was put in place with formalized mechanisms for versioning, deployment, and issue tracking.

From pilot projects two problems arose: first, the need for documentation became more and more evident, then the initial reception from technical analysts was not positive. Technical analysts did not have prior experience with the tools and the solutions they proposed were sometimes not implementable with tools as described by them. Through discussions with developers it was possible to find ways to implement variants of these functionalities in a satisfactory way. However analysts were somehow reluctant to embrace the change, probably because they initially perceived the tools as limiting.

Figure 4.5 shows the ecosystem during this era. The MDD-tools were provided to the software factory, together with basic face-to-face support. Only a few pilot projects were developed using the MDD-tools, they coexisted with projects developed using the traditional (non model-driven) approach. A fundamental change in the ecosystem is that a new type of artifact started to be exchanged: models. Instead of providing the final code, the model was provided with application-specific logic – in protected regions – enabling the generation and re-generation of the final application.

In general the results were considered positive and the organization gain the confidence for a larger adoption of the MDD Tools: during this era the MDD approach grew into a strategic asset for the company.

4.3.5 Maturity Era

Period: *between January 2010 and December 2012.*

Project	Size	Duration From	To	Productivity All	Devel.
Baseline				15	30
β	1091	Oct. 2009	Feb. 2010	32	80
γ	345	Aug. 2009	Dec. 2009	-	48
δ	307	Sep. 2009	Oct. 2009	39	89

Table 4.1. Productivity of pilot projects vs. baseline (function points)

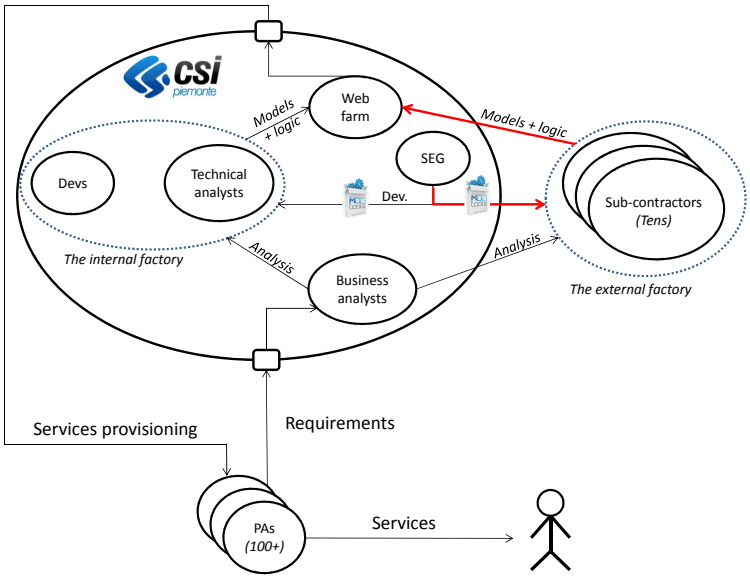

Figure 4.6. Ecosystem during the Maturity era

In this era the MDD approach reaches maturity. Its usage was institutionalized and so it impacted more deeply the organization and started to spread to subcontractors, hence to personnel not directly employed at the company or consulting at the company offices.

Support was offered in a more structured way (see Sect. 4.4.4 for details). In addition, more and more developers, who already gained experience with the MDD-tools, were able to coach novice adopters.

Both *guigen* and *servicegen* were evolved as follows:

- **guigen:** the code generated was checked to be XHTML compliant and standardized. The support for skins was implemented: the mechanism permitted to customize the appearance of generated applications, making easier to adapt

to different PA web portals and to other kinds of customers. A specific group for the development of skins was created at CSI. The tool was evolved to support rich user interfaces, e.g. including GIS maps. A considerable effort was put to guarantee that MDD-Tools generated code compliant with the Italian regulation for accessibility which is particularly stringent for portals for the PA[5].

- **servicegen:** a new generator was added to target the web services framework named Apache CXF[6]. Servicegen permits to define web services and their orchestration. It let the developers model the interface, user groups, authentication rules and so on.

The quality of the generated code was assessed by means of static analysis techniques. In particular the SONAR[7] tools was used and the generator were modified to match internally defined quality thresholds (see Sect. 4.4.6).

The project was released under an open-source license, the EUPL[8]. Opening the project required first the legal team of CSI to review the different candidate licenses and pick up the most suitable one. On the technical side the tools needed to be reviewed to remove elements which were too much "CSI-specific". For example particular services were adopted to implement authentication and authorization of web portals developed by CSI, this aspect and others were modified to be more customizable.

The first efforts to make subcontracting companies to adopt the tool had a limited success: companies were reluctant to invest in a tool which was still considered too much CSI-specific. In particular those companies felt that the investment in training the people for the MDD-tools platform was not enough rewarding.

Anyway, during this era, the internal usage of the tool use grew. At the end of year 2011 more than 200 services were developed using MDD-Tools and more than 70 developers were able to use them autonomously. The growth in the number of users brought upfront the necessity for training, documentation and technical support (see Sect. 4.4.4).

Training was provided by the team who developed the MDD tools through both scheduled internal courses and coaching during the kickstart era of the projects. Finally the documentation was completed and enriched with tutorials, screencasts, and thematic guides.

[5]See Italian law nr. 4 of the 9th of January 2004, and successive modifications

[6]http://cxf.apache.org/

[7]http://www.sonarsource.org

[8]http://en.wikipedia.org/wiki/European_Union_Public_Licence

A communication problem between analysts and developers already emerged during pilot projects; analysts did not understand the nature of the MDD-tools. This brought two types of problem: not only the requirements did not leverage the capabilities of MDD-tools, but also sometimes the requirements turned out to be not practically implementable using the MDD-tools. To solve this issue a showcase project showing the nature of single components was developed. Using it analysts could learn the aspect and the utility of each one of them and started to design solutions referring to these building blocks.

The configuration of the ecosystem at this stage is presented in figure 4.6. MDD-tools are not provided only to the internal factory but also to external sub-contractors, which become part of the new MDD-centered community. The sub-contractors were convinced about the cost-effectiveness of investing in MDD-tools skill acquisition with the promise of a sustained flow of jobs (see motif RoI for adopters in Sect. 4.4.7). At this stage the development rules previously described through documentation are mostly encoded in the MDD-Tools (see automatic enforcement motif in Sect. 4.4.5).

The development is performed using MDD-tools in the whole ecosystem, so the deployment to the web farm is performed by providing the models and the addition logic (in the form of code within protected regions).

Near the end of the community approach a questionnaire was distributed to some of the sub-contractors of the company to evaluate the knowledge of the tool and plan the transition to the community era. Considering that the sample was not built according to designed schema we can not assume absolute representatitivity, however 25 companies were involved. Most of the participants declared interest in learning more about the solutions as users, and a relevant part showed interested also in learning about the internals of the MDD-Tools.

4.3.6 Community era

Period: *since January 2013.*

In this era the economic conjuncture forced CSI to reduce the investments on further development of the MDD-Tools. Nevertheless the company was aware of the necessity of guaranteeing support for existing and new users. At this stage the approach was already largely adopted by CSI and several ongoing projects were using the MDD-Tools. The company decided to focus long term efforts in fostering a stronger involvement of the community in the development of the tools.

In particular the development is not anymore performed by one single central unit; a reorganization deployed a new structure where a single person (*the product leader*) is responsible for managing MDD-tools evolution by receiving contributions from several different business units and teams. While previously some users started

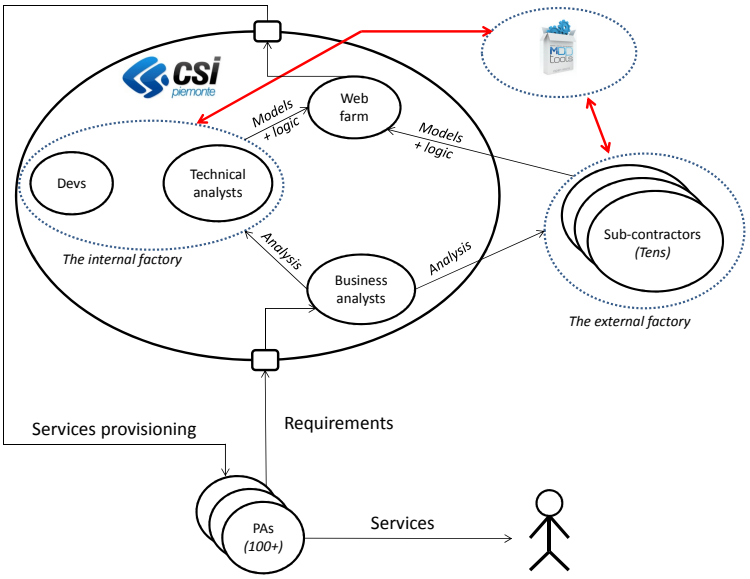

Figure 4.7. Ecosystem during the Community era

naturally to coach and evangelize about MDD-tools this role is now officialized by the company. Moreover those active users are asked to contribute to the development of the tools itself. This change should lead to a tailorization to the needs of the different business units, which will have more control on the evolution of the suite.

Being the tool open-source and given that many subcontractors gained experience using it, it started to be used for projects not involving CSI. Unfortunately precise data on the number of these projects is not available.

The company invested in a course for twelve developers of the community to improve the ability to modify the toolset itself and facilitate new contributions.

In this period different business units contribute to the tools according to internal needs, emerging during the development of specific applications. Contributions are shared and redistributed to the community. A few contributors are starting to solve minor bugs or provide small updates even not directly related to their work, showing ownership of the toolset. For example, a contributor developed a bundle of the toolchain for Linux, while previously CSI developed bundles for Windows and Mac OS X.

Until now the community is mostly self-governed. While each user of the tool is now able to develop functionalities without the burden of synchronization and amministrative overhead on the other side there is the possibility of effort duplication or evolving the tools in different directions. The most active members of the

Motif	Effects on		Era
	Efficiency	Diffusion	
Incremental adoption		✓	Informal
Toolsmithing	✓		Assessment
Integration	✓	✓	Investment
Support		✓	Maturity
Automatic enforcement	✓		Maturity
Generated code quality	✓		Maturity
RoI for adopters		✓	Maturity
Distributed development		✓	Community

Table 4.2. Motifs with main effect and era of appearance

community start to discuss using forums, wikis and a website set-up by CSI; recently some of them are considering the possibility of producing together a new cartridge (a plugin of the generator) to support the bootstrap CSS template[9]. The cartridge previously developed by CSI were designed for the main portals of the customer PAs where the main focus was on accessibility. The adoption of the toolset in a different context requires some adaptation, mainly concerning the layout that should be rendered more responsive and "appealing".

The ecosystem at this stage is depicted in figure 4.7. The MDD-tools open-source project has now become a new stand-alone entity in the ecosystem. The developers in the ecosystem now focus on contributing new features of the MDD-tools, they provide new code to the OSS project. In practice we observe a transition from a centralized development typical of closed-source environment to a distributed development typical of an open-source environment (see Sect. 4.4.8).

4.4 Motifs

The way the ecosystem was shaped by the introduction of the MDD-tools and the way CSI managed to successfully spread over the new paradigm – first internally to the organization, then to the whole ecosystem – are main features we observed in our investigation.

Through different forms of interactions that took place in the last two years – in the context of a collaboration between CSI and Politecnico di Torino – we elicited several of such practices.

The are first of all an abstraction of our interpretation of the evolution of the

[9] `http://getbootstrap.com/css/`

Name	Incremental adoption
Problem	There is resistance to the diffusion of the (development) technology
Context	The diffusion is in its initial phase, from the initial proposers' group to a still small group of potential new adopters
Forces	Tendency to maintain work habits Skepticism about benefits Fear of possible difficulties
Solution	Conduct the adoption of the technology incrementally and on a voluntary base
Resulting context	Word of mouth dissipates fears and skepticism, required knowledge is easily available from the neighbours
Rationale	This approach allows for a spontaneous and persuaded adoption and facilitate the natural emergence of champions

Table 4.3. Motif summary: Incremental adoption

ecosystem and secondly they could turn out to be a reservoir of potentially useful software engineering practices.

We dubbed *"motif"* the most relevant among the elicited software engineering practices. We use the term *motif* because they cannot be considered proper organization [Coplien, 1998] or design patterns [Gamma et al., 1994], both because they are not proved[10] and they cannot always be formulated as solutions.

The motifs we present here, we believe, are peculiar of the evolution of an ecosystem when a new development tool or technology – produced and used by the different members of the ecosystem – is introduced. Their applications – in different eras of the ecosystem history – turned out to be success factors for the favorable evolution and growth of the ecosystem.

We describe each motif using the so called "Canonical form"[11] for pattern description, and we report how the motif was applied in the case under study.

Each motif was applied mostly in one specific era as shown in Table 4.2. They affected two main aspects: the efficiency of the software development and the diffusion of the MDD approach within the ecosystem.

4.4.1 Incremental adoption

Before the initiative started as a formalized task by SEG in 2008, a few tools were already used to perform code generation. Single developers were using tools like

[10]We stress here that our approach is mainly interpretive, we cannot claim a positivist nature for our results, though they could be used as the basis for further a positivist research

[11]`http://c2.com/cgi/wiki?CanonicalForm`

XDoclet[12] or FreeMarker[13]. The former allows generating code from annotations[14] inserted in Java modules, while the latter is a template-based code generator. Because of this personal experimentation some of the developers where familiar with code generation and had experienced its benefits on real projects.

When the SEG introduced *csiskelgen* (i) it was an incremental evolution in respect to previous approaches, and (ii) the adoption was conducted on a voluntary basis. During initial phases developers were free to adopt it or not for new projects, without pressure from project managers or executives. Later the adoption of the tool and its successors spread naturally into different business units thanks to word of mouth and evangelization spontaneously conducted by satisfied users. When the management started to encourage a more structured adoption of MDD-Tools many developers had already either used them personally or heard of success-stories from colleagues.

According to Rogers [Rogers, 2003] in this case a *collective innovation decision* is taken in favor of the adoption of MDD. This situation contrasts with scenarios where the adoption of MDD is forced by the management and the developers of the organization have no prior experience with it. In those cases resistances to adoption are probable [Harrison et al., 1997]. The gradual approach is also suggested in [Hutchinson, 2011].

4.4.2 Toolsmithing

There are several tools in the marketplace that support the development of business application using an MDD approach. During the Assessment Era the SEG carried on an evaluation of the most significant ones.

The result of the evaluation, was that in a large software factory as the one present in CSI the best solution possible seems the development of custom toolchain and structure which could be shaped and adapted to the organization's need. Alternatives based on the adoption of packaged solution were evaluated and discarded in the case of CSI.

Toolsmithing, and in particular the definition of specialized editors seems to be rarely implemented in MDD solutions, with micro companies slightly more inclined to consider it in respect to large companies [Tomassetti et al., 2012]. This is probably due to the fact that more complex processes are typically adopted in the latter, which could be more reluctant to develop their own tools and perform the necessary

[12] `http://xdoclet.codehaus.org/`

[13] `http://freemarker.sourceforge.net/`

[14] `http://en.wikipedia.org/wiki/Java_annotation`

Name	Toolsmithing
Problem	The needs of the company are not met by existing commercial tools
Context	The technological evolution compels a company to adopt new tools
Forces	Existing tools impose their approach The company has a set of constraints that contrast with the tool's approach
Solution	Develop the tool in house
Resulting context	The constraints are satisfied by the new tool
Rationale	Building the tool allows for a perfect customization to the technical and process needs of the company

Table 4.4. Motif summary: Toolsmithing

complimentary steps for a successful deployment (e.g. revise the processes, train the developers).

It is true that a significant effort is required for the development of appropriate technological knowledge to enable such an approach, though this study suggest that practitioners working in software-intensive companies can consider this possibility as a viable option. An industrial survey confirms the role of toolsmithing in improving the chance to achieve improved flexibility, productivity, reactivity to changes and platform independence [Torchiano et al., 2013]. To partially reduce the development cost we also suggest to base custom MDD solutions on enabling platforms like EMF or commercial alternatives (e.g., MetaEdit+).

4.4.3 Integration

The initial tool (*csiskelgen*) was not integrated with the IDE and it was quite unpolished, still it was perceived as valuable from developers.

IDE integration turned out not being critical to promote the first adoption of the tool but it had to be addressed to permit to a larger portion of the users to adopt it.

This turned out to be a successful action in the Investment Era (see section 4.3.4.

The selected technological platform (Eclipse EMF) is well suited to build plugins for the Eclipse IDE that is typically used by the developers in the ecosystem. The integration with the platform permits to benefit of accessory facilities (like plugins for supporting version-control systems) with which the user-base is already familiar. Integration can be seen as a way of mitigate risks and to leverage existing investments [Selic, 2003] both on tools development or acquisition and effort spent on skill development.

Balasubramanian et al. reported missing integration as a problem making difficult to obtain a complexive view of the system [Balasubramanian et al., 2006]. We agree with this finding and we suggest to practitioners to consider tools integration

Name	Integration
Problem	New adopters have problems in using the technology that doesn't fit their usual workflow
Context	The technology starts to be adopted by a larger group not including only enthusiasts
Forces	Developers tend to stick to their workflow Tools implicitly assume a given organization of work
Solution	Integrate the tools into the commonly used development environment (e.g. as plug-ins)
Resulting context	No significant change to the workflow is required to use the tools
Rationale	A least effort solution that allows using the usual IDE without heavily affecting the workflow can potentially overcome the resistance from developers

Table 4.5. Motif summary: Integration

Name	Support
Problem	New adopters have problems in using the technology: they need to train, learn and acquire the specific skills
Context	Scaling up from a small (voluntary) group, through management commitment
Forces	Need for documentation and support Community used direct personal contacts
Solution	Plan of an heavy effort investment in documentation, building a support group
Resulting context	Information is available through the official documentation or resorting to support group
Rationale	When a certain size threshold for the adopted group is reached, nearby colleagues are not anymore available or sufficient for training and problem solving therefore an institutionalized approach to knowledge sharing must be implemented

Table 4.6. Motif summary: Support

when planning the diffusion of technological changes to a large number of different actors.

4.4.4 Support

While for popular and widespread development technologies the knowledge is normally freely available in the web, for tools used only within a certain ecosystem (like MDD-Tools) specific support have to be provided from within the ecosystem itself.

In this case support is needed both during development and maintenance. During development programmers need clarifications about the modeling language and advices on the best practices, during maintenance help is needed for tuning the application and to conduct bug fixing.

Initially the documentation was very limited and there was not a group in charge of offering support, this was due to the nature of the initial incremental and voluntary adoption process (see Sect. 4.4.1 above). At that time users just turned their question to the initial contributors or other developers in the same team.

The transition from in-house usage of the MDD solution to the ecosystem configuration required an adequate management of knowledge. In this respect the situation is similar to what happens with off-shoring: knowledge which existed internally within an organization's boundaries is moved to and exchanged with external organizations [Bahli and Rivard, 2005]. In the CSI case the knowledge considered includes both technical expertise and know-how about processes.

The importance of knowledge transfer is fundamental to properly perform this step and both explicit and tacit knowledge should be considered [Swartbooi, 2010]. In our case-study the explicit knowledge was transmitted through documentation while the tacit knowledge was accessible through the support team and with dedicated training-on-the-job activities.

When the ecosystem grew at a fast pace during the Maturity Era (see section 4.3.6), such approach became infeasible, thus leading to different strategies to guaranteeing support. First the SEG offered both the first and second level support, as the workload kept growing the first level support had to sub-contracted to an external company. A Q&A system (similar to Stackoverflow[15]) was also implemented when the same questions started to appear over and over. The system was made available to all the developers working in the ecosystem.

It is important to emphasize how value can be provided to a small initial circle of users with a limited investment while the leap out of the circle – i.e. delivering value to a wider population of developer – require a much larger investment.

Though, at a later stage, as the tools are more and more adopted the number of advanced users able to offer some support to colleagues grows.

It is important for practitioners to consider both explicit and tacit knowledge and be ready to perform the necessary investments, if they want to fully exploit the benefits of a technological transition.

4.4.5 Automatic enforcement

The control of several different aspects of the applications is fundamental. For instance from a functional perspective the applications must be hosted by the CSI web farm and be integrated with common platform modules, e.g. authentication. From a non-functional point of view, web portals developed for the PA must conform to a set of accessibility standards.

MDD-tools make possible the governance of every aspect of the applications life cycle. Sub-contractors could in principle provide just the models – accompanied by application-specific logic that need to be written manually within the protected regions – to CSI, which then autonomously generates and deploys the application.

[15]http://stackoverflow.com/

Name	Automatic enforcement
Problem	It is difficult to make several different providers (sub-contractors) to comply with a given set of standards
Context	The developed software must comply to legal constraints (e.g., on usability) and technical constraints deriving from the web farm platform
Forces	Conformance to (usability/integrability/standards) rules is required Target platform may vary Developers and sub-contractors tend to adopt most familiar/cheap technologies
Solution	Encode the rules in the code generator
Resulting context	Conformance is guaranteed by the tool
Rationale	Instead of imposing directly the constraints, e.g. through heavy rules and standards books, the constraints are encoded in the tool and the conformance is automatically – and mostly transparently – ensured by the everyday working tools, e.g. by generating conforming code

Table 4.7. Motif summary: Automatic enforcement

This approach permits the enforcement of standards and procedures with a limited effort.

Upon receiving the models, CSI has the option to check them to verify that quality requirements are met, for instance the rules for usability could be mostly automatically verified. This is made possible because an high level of abstractions is adopted (models instead of code). In addition it is possible to use a standardized building process to generate the code and produce the final application. Finally, the deployment also can be performed in a standardized way.

The transition to the MDD-based ecosystem with automatic enforcement during the Maturity Era (see section 4.3.6), brought significant benefits. This approach contrasts with the reality in other contexts – e.g. PAs in close regions – which experience strong difficulties in managing, installing and maintaining a plethora of interdependent applications developed with different technologies, on different platforms and with different requirements.

The idea of enforcing architectural rules was envisioned before [Mattsson, 2008]. Normally it is executed as a supplementary step in which architectural rules are automatically or manually verified [Mattsson, 2010]. In this case rules were not anymore formalized explicitly, but they were embedded in the generators and the solution itself.

We suggest practitioners to consider the benefits of automatic-enforcement in managing relations across an ecosystems. By reducing the cost of evaluation and making more objective the process, the level of confidence in the relation can grow.

Name	Generated code quality
Problem	The poor quality of generated code affects the overall quality of the product
Context	The development tool produces code, that will be shipped as part of the final product, e.g. using an MDD approach
Forces	Quality of generated code is rarely put into question Generated code has the potential to cripple the whole application Generated code could need to be understood, e.g. to integrate, debug, troubleshoot Generated code is not intended for developers to read
Solution	Assess the quality of generated code and improve the generator
Resulting context	The final quality of the generated code is improved, which is reflected in improved overall quality of the product. In addition the understandability of the code is improved
Rationale	Several quality issues may indicate bugs that otherwise would be difficult to understand, and the resulting improved quality often implies better understandability

Table 4.8. Motif summary: Generated code quality

4.4.6 Quality of the generated code

During the Maturity Era (see section 4.3.6), the SEG performed an investigation for quality issues on the generated code and individuated a few actual bugs. It is important to notice how an investment on the quality of the generated code could be immediately retrofitted to affect all the projects developed using the code-generation feature of the platform.

Normally the quality of generated code is not monitored; this case suggest it could make sense to do that because:

- the generated code could need to be integrated with custom code written by developers and not generated,
- also if manually written code is not inserted the generated code could still need to be read and understood for debugging and troubleshooting,
- quality problems in the generated code could lead to errors or performance issues.

Investigation on the quality of the generated code is often neglected. We suggest it useful to improve two factors: performances and code readability. For most applications the performances are simply not an issue, while for applications where performances are critical MDD tend to be rejected to retain full control of the executed code [Selic, 2003]. Code readability of generated code is normally not much considered: the best-practices suggest that all code should be generated [Kelly and Tolvanen, 2008] and considered as a semi-artifact necessary just to obtain the compiled applications. We argue it is not frequently the case: in many MDD solutions

Name	RoI for new adopters
Problem	New adopters are reluctant to invest in training on new technology
Context	The development tools requires a non-negligible effort in training and learning, while the technology is mainly used within the ecosystem
Forces	Large upfront training investment Limited reusability of the new skills
Solution	Provide a medium-/long-term commitment in providing jobs
Resulting context	Upfront investment is repaid by long-term job series Acceptance at company level may trigger a similar problem at individual level
Rationale	The new adopters (sub-contractors) must have some form of guarantee that the investment yields a return through a long-term job agreement

Table 4.9. Motif summary: RoI for adopters

a large fraction of code is generated but corner-cases are managed through hand-written code [Torchiano et al., 2013] which need to be integrated with generated code. In this scenario the readability of generated code becomes important.

Investments in the quality of the generated code can be incremental and benefit not only applications currently being developed but also past ones – by means of a regeneration of the code – and future ones. On the contrary investments on the manually written code of one applications can benefit only the individual application being targeted.

Working on improving the quality of code, previous findings should be considered: previous research [Vetro' et al., 2011, Vetro' et al., 2013, Vetro' et al., 2010] indicates that while a portion of warnings issued by static analysis tools are highly correlated to potential errors, most of them are not. The latter type of indicators cause an high workload corresponding to a small or null contribution to the improvement of quality. Therefore when evaluating the quality of the generated code the relevant indicators need to be identified using techniques which require competencies on the technologies involved and on statistics.

4.4.7 RoI for external adopters

The adoption of the MDD-tools by sub-contractors required, on their side, a significant investment in acquiring the specific skills for using that platform. Such knowledge, though, could only be profitable when developing for CSI. During the Maturity Era (see section 4.3.6), the decision of a sub-contractor to adopt the MDD-tools therefore consists in evaluating the return yielded by such an investment.

An affordable return on investment is possible if CSI is able to guarantee a given amount of work to be sub-contracted. Once sub-contractors felt that there was a sufficient yield they were willing to invest in training for MDD-tools, with the possibility of using it also for other customers.

Name	Distributed development
Problem	Centralized development team can hardly identify direction for enhancing the tools
Context	A centralized team manages the development a fairly mature technology
Forces	Difficult to identify directions for evolution Lack of knowledge about the specific Need to evolve the technology
Solution	Distribute the development to the individual business units that use the technology
Resulting context	The needs are address by evolution within the same context where they arise
Rationale	The evolution is pushed and driven by specific needs, plugged into a stable and mature core

Table 4.10. Motif summary: Distributed development

While this strategy eventually led most sub-contracting companies to decide for the adoption of MDD-tools, individual developers in such companies were sometimes still reluctant in investing too much in learning the MDD-Tools. Their fear was to reduce their ability to find a different job, because the skills acquired with MDD-Tools are relevant only inside the ecosystem of companies using them. Developers would prefer to invest in skills which provide access to a larger job-market (e.g., java programming). In this case the advantage of the business unit or the benefit of the supplier need to be put in front of the prospective of the single developer. On the other end developers using MDD-Tools slightly change their role and develop a different skill-set which could be useful to work with other higher-level DSLs. To individuals to benefit of this acquired skills the industry need to adopt them more largely.

4.4.8 Distributed platform development

Since the MDD-tools platform reached its maturity, further evolution mainly address new needs that emerge in different domains during the development. The central team – the SEG at CSI – in charge of the tools' evolution encountered several difficulties in collecting, understanding and implementing the requirements coming from the development teams.

Within a large ecosystem, not only the evolution becomes critical, offering a centralized tool support shows some drawbacks: the person offering support typically does not know the project and she is not aware of its peculiarities and does not feel involved in the project. In addition, this division introduces an attribution problem [Jaspars et al., 1983]: the people in central support unit tend to be considered responsible for most problems even loosely linked to the MDD-tools and they feel they are blamed too often, while developers working in the other business units tend not to considered themselves fully responsible for the correct implementation of their projects.

In an attempt to solve the above problems the tools' evolution was later moved to a distributed schema: individual business units are in charge of developing the extensions they need. External partners are welcome as well to provide contributions. Most of the people originally working in the support unit have been relocated in the business units, while at the central level is maintained only the role of coordinating the development of the tools, to maintain a cohesive strategy and to avoid duplication.

4.5 Discussion

The whole history of the ecosystem evolution represents a continuous improvement both for the collective and the individuals from several perspectives. The final configuration of the ecosystem is an improvement in respect to the initial situation for a variety of reasons. Here we summarize the changes occurred in the ecosystem and its evolution along a few relevant dimensions.

Productivity: the development process is based on more productive tools, which permit to consistently reduce the development cost and provide more predictable development time. The productivity measured in the Investment era (see Table 4.1) indicates an improvement factor ranging from 1.5 to 3, w.r.t. the baseline. This result is consistent with a previous survey of modeling and MDD in the Italian industry [Torchiano et al., 2013].

Skills: the ecosystem allowed many companies of the regional IT district to acquire competencies on MDD which were scarce before. As MDD-Tools are adopted by sub-contractors not for new customers, different from CSI, the competencies developed on MDD-Tools become more useful and led more practitioners to learn about MDD. Actually the lack of competencies has been reported among the top three reasons why MDD techniques are not adopted in the Italian industry [Torchiano et al., 2013]; we expect the new skill will trigger a virtuous circle that will lead to a wider adoption of such techniques in the area.

Control: before the adoption of MDD-Tools, manual code inspections were necessary to verify the adherence of applications to the company development standards. After the adoption of MDD-Tools a large portion of code is automatically generated from models and therefore the generation process automatically ensure the strict adherence to those standards. In summary MDD-Tools significantly simplified the governance of the applications. Transitions to new platforms are now possible and cheaper, because the technological aspects are captured only into the code generators, instead of being spread through all the codebase. This permits to migrate applications as the implementations

become obsolete, preserving the investment done by customers. For example a transition from MySQL to PostgreSQL in the CSI web farm was performed during the period of interest: thanks to the generative approach it required to adapt only the generators and it was possible to apply it to a large number of applications with a limited effort.

In the future it could be possible for CSI to acquire directly and exclusively models which are easier to verify. Some experimentation on metrics for MDD-Tools models are already started. They could be an important tool for the final customer to evaluate the quality of the applications received and to permit a fairer competition between different suppliers.

Flexibility: In a software ecosystem involving tens of companies, as the one centered on CSI, there are several specific needs and constraints, most of which percolate from the customers – i.e. several PAs –. During the *Assessment* era (see Sect. 4.3.3) a survey and assessment of commercial packaged solutions was conducted; they were discarded because of lack of flexibility.

The MDD-tools were instead developed as custom tools and structure which could be personalized using a DSL, building upon the enabling platform of EMF. Such a solution proved itself flexible enough to accommodate all the different development needs that emerged in the ecosystem. Toolsmithing represented a success factor, confirming what emerged from a previous survey [Torchiano et al., 2013]. The necessity to be able to craft their own tools seem fundamental for software intensive companies of each size, as emerged also in a case-study previously conducted by authors of this work [Tomassetti et al., 2013a].

Roles: Traditionally two figures constitutes the bulk of the employees of the internal and the external factory: technical analysts and developers. With the switch to a MDD paradigm of development these factories need to produce mainly models and the accompanying logic (still written using general purpose languages).

Technical analysts are reluctant to model because it means providing a technical artifact which is used to directly produce the concrete applications therefore more responsibilities are involved. While errors in the technical documentation – e.g. word files – currently do not lead to dire consequences. On the other end the developers do not want to just model because the related skills are less useful – in terms of their career – in other contexts than the knowledge e.g. of some mainstream programming language.

Currently models are written mainly by people trained as software developers. In the long run the change operated in the ecosystem with the adoption of

MDD call for a redefinition of the figures involved in the software development because the skill set required for writing models using MDD-Tools seems slightly different from the one of developers and technical analysts. The optimal solution seems the definition of a new role for which at the moment there is an absolute scarcity in the ecosystem. Modellers are a new intermediate figure. They could be seen as designers who could benefit from some knowledge about user-experience design principles and usability.

Given the number of personnel employed in the ecosystem it is a need that could affect the regional market job and should be considered also from the local universities.

4.6 Related work

The related work is organized in two parts; first we consider the problems and guidelines in deploying MDD and Software Product Lines (SPLs), then we focus on software ecosystems.

4.6.1 Deployment of MDD and SPLs

Baker et al. [Baker et al., 2005] describe the effects of the adoption of MDD in a large company (Motorola) along 15 years. In their experience the major obstacles in adopting MDE stem to the lack of a well-defined process, lack of necessary skills and inflexibility in changing the existing culture. In CSI there were already well established processes and the company was able to form the necessary skills along the years. Another account from a 5-years project at the same company can be found in [Foustok, 2007]. The author reports as difficulties the ability to cope with an increasing rate of change in the technology. In the case of CSI they were strict about dictating transitions to new technologies and they were not forced by customers to adopt particular versions or technologies. An important difference in the two experiences is the usage of UML at Motorola, while at CSI a set of DSLs was developed.

Fleurey et al. [Fleurey et al., 2007] report about a case-study of MDE adoption. The case-study considered a time window of 10-years; the focus is on migration projects, where the benefits w.r.t. conventional techniques can be observed after an initial period, e.g. the first code could be delivered only after 10 months from project's beginning. In addition they present a cost-benefit analysis and suggest the presence of profitability threshold in terms of project size. The authors suggest that domain applications last longer than technologies, which change at a faster pace. Also in the case of CSI MDD, the key features are guaranteeing a longer life

to applications and facilitating the unavoidable transitions to new technologies – which are operated updating the generator and regenerating all the applications –.

While more focused on small companies, authors of this work examined the best practices for deploying MDD applications [Tomassetti et al., 2013a]. Some lessons learned for small companies apply also to this case: i) the necessity of flexibility, to achieve using DSLs and customized solution instead of framework or products off-the-shelf, ii) the importance of buying the developers committment. The first point is critical also for CSI, we suggest that software intensive companies need flexibility, to be able to tailor the development processes considering the competencies and the specificity of the company. Organizational aspects and the importance of taking in consideration the reactions of external partners are instead peculiar to large organizations, given the impossibility for small companies to shape external factors. On one hand it means small companies have a more rigid environment to adapt to, on the other hand they have to consider less aspects in designing their MDD solution. Given that the number of users is not likely to grow up beyond the organization boundaries the MDD can be deployed with a limited investment (as it was in the first phases of the CSI case-study).

Hutchinson et al. in [Hutchinson et al., 2011b] report the results of an empirical study on the assessment of MDE in industry. Their work has two goals: identify the reasons of success or failure of MDE and understand how MDE is actually applied in industry. They employed three forms of investigation: questionnaires, interviews, and on site observations, having as target practitioners, MDE professionals and companies practising MDE respectively. The questionnaire has received over 250 responses from professionals (the most of them are working in Europe). Some of the reported findings are: (i) about two-thirds of the respondents believe that using MDE is advantageous in terms of productivity, maintability and portability (increase productivty was verified in the case of CSI, as well as portability), (ii) the majority of respondents use UML as modelling language, and a good number use in-house developed DSLs (the latter was the choice of CSI), (iii) almost three quarters of respondents think that an extra training is necessary to use MDE (we have seen that CSI invested in training), (iv) the majority of respondents agree that code generation is an important aspect of MDE productivity gain (code generation was also the choice of CSI), and (v) a little less than half of the respondents think that MDE tools are too expensive (we can confirm that CSI had to invest consistenly to create the MDD-Tools). We observed similar perceptions in a survey conducted by us [Torchiano et al., 2013] except for the issue of extra-training which was not considered in our survey, however we observed that the lack of competencies is one of the problems most frequently reported by companies. Differently from the results of their survey, the cost of supporting tools is seen as a problem only by a small proportion of respondents in our sample. Probably it depends on the choice to use existing tools or develop them, and on the size of the user base: we have seen that

in the case of CSI a small user-base was able to use profitably an immature toolset, while it had to be considerably evolved to be adopted by a large user-base.

Hutchinson et al. [Hutchinson et al., 2011a] report lesson learned from adoption of MDE in three large multinational companies (a printer company, a vehicle manufacturer and a manufacturer of electronic systems). In particular, the importance of complex organizational, managerial and social factors in the success or failure of the MDE deployment. The authors report some organizational factors that can affect the success or the failure of MDE deployment. The factors that can affect it positively are: (i) a progressive and iterative approach, (ii) user motivation in the MDE approach, (iii) an organizational willingness in integrating MDE in the whole organization, and (iv) having a clear business focus (where MDE is adopted as a solution for new projects). Instead, factors that can affect it negatively are: (i) the decision of adopting MDE being taken by IT managers, in top-down fashion and implemented "all at once" without developing gradually an understanding of the necessaru process changes, (ii) MDE being imposed on the developers without providing the right motivations, and (iii) an inflexible organization with a lack of integration of MDE in previous processes. In the CSI case all the positive factors were verified and no one of the negative ones, leading to an ideal situation. The only common aspect with the work proposed in [Hutchinson et al., 2011a] concerns the motivation of developers.

Mohagheghi et al. [Mohagheghi et al., 2012] interviewed – using convenience sampling – developers from four companies involved in an initiative called MODELPLEX. They examined the factors affecting adoption of MDE. Regarding usefulness they found uncertain results: most participants recognize the usefulness of models but they are not sure about the impact on the quality of the final product or the effects on productivity. MDE is perceived as not simple: its complexity makes it viable for engineers but not for non technical people. This finding is confirmed by our results reported in [Torchiano et al., 2011b, Tomassetti et al., 2012]. They show that only in a few cases business experts are involved during modelling tasks. Also in our case-study the introduction of MDD led to re-consider the different roles and their involvement in the development phases (see Sect. 7.6 about Roles).

Regarding compatibility with the existing development process the companies complained about the lack of standards and the consequent lock-in effect. All interviewed companies reported some problems in integrating their existing approaches with MDE. Tools could have been part of their problems, them being not considered satisfying by a part of the sample. In particular, some participants expressed several concerns about the scalability of the MDE approach to large projects (this could be related to the motifs *Integration* and *Support*). Advantages reported are limited to the usefulness for documentation and communication purposes. Major reasons preventing adoption of MDE are the immaturity of tools and processes as well as the lack of competencies.

Catal [Catal, 2009] discusses some of the barriers to the adoption of Software Product Line Engineering (SPLE). Among the other findindgs, the author suggest to consider SPLE as mean to obtain resusability at different levels: not only of implementation artifacts, but also of documentation, tests, practictes and other complimentary elements. According to Catal some of the problems derive from an unclear terminilogy (with duplicate terms used in US and Europe) and a lack of resources to learn how to implement SPLEs. The necessity of deep changes in the organization's process make seniors to refrain and resist the migration.

Authors of this work partecipated on an industrial survey about MDD adoption in Italian companies [Torchiano et al., 2013]. Among other findings, from our survey emerges that the most common problems in deploying MDD are the size of the effort required, the necessity to prove the usefulness of the solution designed, the lack of competencies and proper tools, the missing support from management. In this case the company solved this issues through a slow evolution which made possible to limit the initial investment and to progressively buy-in management and developers support. The size of the company made possible to face the investments necessary both to design the solution but also to grow the necessary competencies. We think that this partial transfer of competencies to external companies and the availability of the toolset to other companies could help other companies part of the district to adopt MDD, lowering the entry-barriers.

4.6.2 Software ecosystems

In 2009 Jansen et al. [Jansen et al., 2009] proposed a research agenda for software ecosystems (SECOs). The three prospectives considered are the *software ecosystem level*, the *software supply network level* and the *software vendor level*. On the *software ecosystem level* strategies are implemented to keep the ecosystem profitable for all the participating actors; our motif *RoI for adopters* clearly operates at this level, also all the motifs related to efficiency could be considered as related to this level because the efficiency of the tools positevely affect all the participants in the ecosystem. On the *software supply network level* relations with suppliers and buyers are considered: orchestration between partners at different part in the process-chain is administered at this level; the motif *automatic enforcement* permits to reduce the cost of verifying the conformity of the products with the requirements, an important aspect in managing the relations with suppliers. Finally at the *software vendor level* actors considers the effect of the SECO on their own products catalog. For each prospectives different challenges are reported. Our work addresses a challenge at the *software ecosystem level*, in particular *Developing policies and strategies within SECOs for SECO orchestration*.

Barbosa and Alves [Barbosa and Alves,] presented a systemic mapping study on SECOs. Authors included 44 relevant papers. The study, being conducted in 2011,

include papers up to year 2010. The number of papers show an increase of interest in SECO in the most recent years considered (2010-2011). Of these 44 papers 4 are focusing on SECO and SPLs; it seems to suggest that more research is needed in this direction, and this case apply to MDD-Tools.

Angeren et al. [van Angeren et al., 2011] performed a survey in the Dutch software industry about the ecosystems companies are working in. Data was collected from bachelor students according to given schemas. In the end the data of 17 companies was considered. Authors individuate four different categories of components, obtained from the combination of two dimensions (critical/non critical, core/contextual). For each category they report which factors influence the relations with suppliers. Factors are: level of intimacy, continuity, visibility within the marked, niche creation, product & license type, support & maintenance. According to their schema the MDD-Tools could be considered as a critical core component, therefore all the factors would be relevant.

Bosch [Bosch, 2009] examines how successful software product lines can evolve to large ecosystems. He considers in particular the case in which a company starts using a SPL for developing its own products and later open it to other actors. The SPL evolve into a software ecosystem when the platform starts to be used outside the organization boundaries. According to Bosch the two main reasons motivating the transactions are: i) the impossibility for the company to sustain alone the R&D costs, ii) the mass customization necessary in certain sectors (e.g., web) for application of the SPL to different customers. In the case of MDD-Tools the major advantages guiding the transitions were the need for standardization of development times, cost and platforms, together with an easier governance of outsourced components. Bosch proposes also a taxonomy of software ecosystems, considering the category (end-user programming, application, OS) and the platform (desktop, web, mobile). We would suggest to add the "tools based ecosystem" which is somewhat similar to the end-user programming ecosystem presented by Bosch (in both case the central element is the tool used to develop applications) but some aspects are fundamentally different: in particular "end-user programming" ecosystems are based on the fact that few investments are needed for the adoption while out suggested category (tools-based) would imply relevant cost for teaching and supporting the developers adopting the tool. In both cases the deep customization of the applications is based on DSLs.

Hannsen [Hanssen, 2012], similarly to us, studied the evolution of a Software-Product-Line centered ecosystem for a period of five years. While in their case-study the ecosystem is born in the period of interest and the SPL was adopted, in our case-study both the SPL and the ecosystem were there from the beginning; however in the period of interest there was a critical technological change in the implementation of the SPL. The company considered by Hansen is smaller than CSI (260 people employed against 1200) but it is a worlwide distributed company, instead of a locally based company as CSI. The author describes in details the

involvement of the customers as an important aspect of the ecosystem. This strong attention to stakeholders seem to derive from the transition to the Evo method [Gilb and Brodie, 2005]. Engaging customers became part of the culture of the company, with reflections at different levels. While initially it required an active effort to involve stakeholders, later the company was able to trigger a strong interest and constitute a pool of very active partners. The company was also able to act as a catalyst for the 60 external organizations which are basing their business of the product line. They did it organizing conferences, opening a web portal and nurtuting the network of partners; some of these ideas could be applied by CSI in the future to strengthen the network of MDD-Tools' users. This is particularly good for those external organizations because they can easily get in touch with a large pool of established users. An efficient integration technology was considered an enabled for this set of extensions, customizations and interoperable solutions. The author presents also a set of theoretical propositions defining a software ecosystem. Among them we note i) the presence of a central referent organization, acting as hub of the software ecosystem. In our case CSI is clearly playing this role; ii) the enabling role of a particular technology, in our case study MDD; iii) the shift to a shared responsibility model on the development and control of the ecosystem, this element is present also in our case-study with the release of MDD-Tools as an open-source platform.

Manikas and Hansen [Manikas and Hansen, 2013] conducted a systematic review of the literature about software ecosystems (SECOs). Authors based their method on the guidelines from Kitchenham and Charters [Kitchenham and Charters, 2007]. They considered 420 papers and included 90. Here we report a summary of their findings: i) there are different definitions of software ecosystem being used, the ones most widely referred come from Jansen et al. [Jansen et al., 2009] or Bosch et. al [Bosch, 2009, Bosch and Bosch-Sijtsema, 2010b, Bosch and Bosch-Sijtsema, 2010a]; ii) the number of papers published on the topic is increasing significantly, raising from 3 papers published in 2007 and 2008 to the 32 published in 2010 and 2011 (the papers' extraction was performed during June 2012); iii) almost half of the papers are reports, while very few use empirical methods. This is an aspect common to many other fields of the SE; iv) papers have a sort of equally distributed focus between SE, business and management, and ecosystems relationships; v) half of the papers refer to an existing SECO.

Bosch and Bosch-Sijtsema discuss about the impact of software product lines and ecosystems [Bosch and Bosch-Sijtsema, 2010b]. They suggest that large-scale software development is hindered by a too much integration-centric approach while switching to a composition-oriented approach would significantly simplify it. To reach this goal the factors motivating strong and close interconnections are examined in this work. Initially authors present three trends affecting large software development (SPLs, global development and SECOs). Based on their experience

and in particular on three case study companies authors present problems common to software intensive companies. Some of the problems derive from the software architectures, making costly the integration and difficult the indipendent evolution of parts of the system. Other problems could derive by engineering practices and from the R&D organization. Finally authors present five approaches to facilitate the transition to a more composition-oriented system: i) consider integration during development, ii) release groupings, iii) release traits, iv) independent deployment, and v) open ecosystem.

Lungu et al. [Lungu et al., 2010] presented a tool to visualize the evolution of SECOs. The tools is named Small Project Observatory and it is an online visualization tool focusing on super-repositories (federation of repositories). It could be used as a supporting tool in studying the patterns of evolutions of applications and improve reusability through the definition of processes common to the ecosystem. That would help to pose the basis for a transition to more structured approaches, as the one proposed by CSI.

Kilamo et al. [Kilamo et al., 2012] list a set of guidelines for successfully release as open-source a proprietary software system and grow a proper ecosystem. Authors derived those guidelines from applying a previously depicted process (the OSCOMM process framework) to four different case-studies. The OSCOMM framework consists of three phases: i) an evaluation of the readiness of the project for being opened, ii) open source engineering the product, and, iii) measuring the ecosystem once the project is open.

4.7 Summary

In this chapter we summarized the five years long evolution of a software ecosystem centered on a large IT organization (CSI-Piemonte). In particular we presented eight motifs that played a significant role in the successful deployment of a paradigm change in such a complex ecosystem.

Although the motifs are derived from one single study – which is a limitation to their validity –, this particular study spanned across many years and considered hundreds of projects, we therefore think that the distilled motifs represent a valuable contribution for deriving best practices in performing paradigm shifting within a software ecosystem.

Some of the motifs presented played a role in creating an *efficient* ecosystem, while other were important to favor the *diffusion* of the paradigm change. Some of them played a role in both aspects. The diffusion was initially favoured by *Incremental adoption* on voluntary basis, then offering the necessary *Support* and *Integration* in the toolchain. In the later eras complementary aspects had to be considered like a proper *RoI for adopters* and a shared responsibility for the solution, obtained

through *Distributed development*. The diffusion was also indirectly favored by realizing an efficient ecosystem first of all with the fundamental choice of performing *Toolsmithing* (with all the implications at an ecosystem level). *Integration* could be also considered an element which brought to an efficient solution. Finally investments in *Automatic enforcement* and *Generated code quality* were particularly rewarding (and sustainable) thanks to the economies of scale of the adoption at an ecosystem level.

With this interpretative work we attempt to propose key motifs for a paradigm shit in a software ecosystem. We believe more interpretative works are needed in the area, to find similarities and variabilities with analogous evolutions in other ecosystems. At that stage generalization would be possible by analyzing the different single experiences.

In particular we believe it would be useful to compare this work with other regional software development ecosystem studying the impact of introducing a rarely used paradigm to a large number of developers in a specific area. The effects in terms of know-how diffusion, the number of initiatives which independently spread out from this effort have yet to be evaluated. We believe that local ecosystems can lead to an unexpected number of interactions which have to be fully studied and understood.

4.8 Modeling adoption: comparison between small and large companies

While the technical challenges in the adoption of MDE are the same, the availability of resources and the internal organization are very different between small and large companies [Richardson and von Wangenheim, 2007], therefore the strategies for adoption should consider this aspect and be planned accordingly.

The first difference is in the rate of adoption: according to the results of our survey small companies seem to use less frequently MDE in respect to large companies. This is reflected also by the literature, where very few case studies involving small companies are produced in respect to those involving large companies. While most of works focus on large companies a few are available also on small companies. In this section we try to summarize some of the differences we emerged, considering both our experience and relevant resources.

Completeness: Cuadrado et al. present their experience in two transfer of technology projects on two small companies [Cuadrado et al., 2013]. They suggest small companies should use MDE to automatize some development activities, while not shifting completely their prospective from code-centric to model-centric development. This is due to the high cost which are associated with the shift. Large

companies can instead consider the transition to model-centric development because they can afford the consequent costs in deep processes re-arrangements.

Technical vs organizational aspects: in our work and in the work from Cuadrado et al. as well, emerge that small companies have to focus on the technical excellence of the MDE solution; it has to be a competitive advantage, which can be sustuainably maintained. In both cases authors underlined the necessity of model-to-model transformations and intermediate model. In the case of large companies the focus seem instead on the completeness of the solution proposed: for the company to be efficient a large set of supporting tools and organizational aspects have to be considered. Repositories for models, integrated toolchains, code checkers, all of them need to be realized and combined to a offer a fully complete solution.

Investments: small companies are budget-driven, they have to deal with limited time and monetary resources. As consequence they need to produce results as soon as possible, therefore solutions are typically deployed in an incremental way. The choice of good pilot projecets is crucial: small companies can not affort the costs of a long trial-and-error process. Large companies can instead plan the transictions using more time, because they typically have the resources to sustain long-term investments.

Flexibility: in general small companies are more ready to deal with changes [Richardson and von Wangenheim, 2007]. Employees of small companies seem more prone to deal with uncomplete or partial solutions. Being flexibility generally regarded as an advantage to maintain many perceived risks on the adoption of MDE are related to the fear of losing flexibility to due to the constraints imposed by MDE. This fear seems less common in large organization which are probably more accustomed with formalized procedures.

Chapter 5

Cross-language interactions: a classification

In this Chapter we work towards answering to **RQ B.1**. Here we introduce the concept of cross-language interactions; we explain what they are and we present a classification schema. Most of the content of this chapter is based on our paper presented at ESEM 2013 [Tomassetti et al., 2013b]. That was a joint work with Antonio Vetrò and Marco Torchiano.

5.1 Introduction

Polyglotism is largely recognized as an almost ubiquitous characteristic of modern software development projects: they use several different languages [Wampler et al., 2010]. For instance, most trivial web applications are typically written in a general-purpose language, e.g. *Java*, include some *SQL* queries, are visually presented by means of *HTML*, formatted using *CSS* files, and with client-side processing implemented using *Javascript*. Another case is the use of Domain Specific Languages (DSLs) that are quite common when adopting a model-driven approach [Torchiano et al., 2013].

An important side effect of polyglotism within a single project is the interaction between languages. From previous studies [Vetro' et al., 2012] [Tomassetti et al., 2013c] we learned that the majority of commits in open source projects are cross-language, i.e. they involve files written in different languages.

Identifying the interactions between artifacts in different languages (cross-language) is important because most development environments, with the notable exception of some platform-specific IDEs – e.g. some Android IDEs – do not provide any support for managing them. A good knowledge of cross-language interactions is a key factor in building a specific support into IDEs with the goal of supporting

development, maintenance, and comprehension activities on polyglot applications.

Two main approaches to language identification are possible. Logical interactions [Gall et al., 1998] occur when two artifacts are modified in the same commit. Semantic interactions occur when within an artifact we can find some elements that link it to another artifact.

While the issue of language interaction is already very relevant today, the appearance of language workbenches [Fowler and Parsons, 2011] let us suppose that this issue is going to become even more important in the future. For example, with Xtext [Eysholdt and Behrens, 2010] and GMF [Seehusen and Stølen, 2011] we can create, textual and graphical DSLs with custom editors integrated in the Eclipse platform with a minimal effort. Other tools like Intentional Software [Simonyi et al., 2006] and the Meta-Programming System [Völter, 2011a] fully support the Language Oriented Programming paradigm [Dmitriev, 2004] and are based on projectional editing. The existence of these tools and their usage in industrial projects [Völter and Visser, 2010] seem to indicate that the interaction between languages in projects will increase in the future.

Our hypothesis is that in the long run we need to support cross language development, including design, modeling, and validation. To reach this goal we first need to start understanding the effects of languages interaction: this work is intended as a first step in that direction.

5.2 Related work

Gall et al. [Gall et al., 1998] proposed a definition of logical coupling between files based on the observation of a software repository. They defined as logically coupled two files which changed together in at least one commit. This allows identifying possible relations which cannot be easily found with a more rigorous syntactic analysis. Another advantage of this approach is the possibility to apply it to all possible kind of artifacts. In later work [Ratzinger et al., 2005, Ratzinger et al., 2007], Ratzinger et al. showed that logic couplings defined on the basis of a repository's history could be used to find artifacts which need to be refactored (reducing the coupling). This is a complement to syntactic coupling.

Mayer and Schroeder [Mayer and Schroeder, 2012] name the problems of references across artifacts written in different languages as "semantic cross-language links". Being these links out of scope of the individual programming language, they are ignored by most language-specific tools and are often checked only at runtime. They propose to explicitly express constraints for these links and present three possible approaches to do that: at source code level, using language-specific meta-models, and using language-spanning meta-models.

5.3 Method

We devised a research method to achieve the dual objective of identifying interaction categories and classifying the occurrences of interactions in a real project.

The procedure we followed is made up of the six steps below:

1. *screening*: we identified logical interactions by selecting the *cross-language* commits using the approach based on file extensions, defined in [Vetro' et al., 2012].

   We adopted this approach to focus on a limited number of candidate pairs since examining all possible connections between every pair of files in a large project requires both a deep knowledge of the project itself and a huge effort.

2. *commit selection*: we selected the bug-fixing commits from the project version control system.

   This choice is motivated by the fact that they typically represent a focused modifications involving a limited set of files. Considering that we are interested in binary relations between files, a large number of files could lead to an exponential number of possible pairs of files to be analyzed.

3. *manual verification*: we verified the language of the files to confirm the presence of *cross-language* logical interactions in a predefined temporal range (the same used in the previous study).

   Since the previous step is based on the file extensions alone, some false positive are possible. Where different extensions actually correspond to the same language or viceversa the same extension (or lack of) correspond to different languages.

   This manual inspection led us to classify as bash scripts files that had not an extension. In a few cases that left us with a commit where only bash files were modified, therefore the commit was clearly not a cross-language commit and so it was excluded from further examination.

4. *semantic interaction manual confirmation*: we manually inspected the files modified simultaneously in the same commit, using mainly the contextual diffs of the involved files and the relative log message to identify *cross-language* interactions and to assign them to a class.

   In this way we progressively constructed a taxonomy of semantic interactions.

5. *revision of the classification*: we discussed the classification built in the previous step, merged similar categories, and defined more meaningful labels.

The goal of this step is to come up with a clear and precise definition of the *cross-language* interaction categories and provide representative examples. The results at this stage are presented in section 5.4.

6. *semantic interaction classification*: we re-processed all the commits and performed a definitive classification of the cross-language interactions according to the final taxonomy.

   When several instances of the same relation were found between the same pair of files (e.g. many *Shared ID*) just one occurrence was reported. Though the same relation could possibly be counted more than once, if it appears in distinct commits.

   The result at this final stage is a set of *cross-language* semantic interactions identified over a set of commits. We then conducted an analysis – presented in section 5.5 – of such data aimed to: i) verify the precision of the logic interaction approach in terms of semantic interactions, ii) define a frequency profile of *cross-language* interaction categories.

5.4 Categories

As a result of step 5) in the procedure delineated in section 7.4, we built a taxonomy of the cross-language semantic interactions. The interactions between different languages can occur in several different forms. While it is extremely common to use more than one language in a single project, it could even be the case that different languages are used in the same file. For instance consider the presence of an utterance of SQL embedded in a valid expression of an host language (typically a General Purpose Language like Java or PHP) or the preprocessing languages as the C preprocessor language or M4. In our work we assumed that it is possible to identify a main or host language in which a certain artifact (e.g., a file) is expressed. We decided to focus only on interactions between distinct artifacts taking into consideration the main language of each file.

We emphasize that the identified categories are not mutually exclusive. For example a Java class could load an XML file (*Data loading* relation) and then perform some processing on specific part of it, using identifiers for the navigation. In that case normally the same identifier is present both in the Java and in the XML file (*Shared ID* relation). In our example therefore there will be both *Data loading* and *Shared ID* relations on the same pair of files.

In Table 5.1 we report the definition of the categories we identified. In the rest of the section we present an example for each category. Examples are derived from the interactions classified according to the procedure in section 7.4.

Table 5.1. Categories for the implementation of language interactions among different artifacts

Category	Definition
Shared ID	The same ID is used among the artifacts involved in the interaction.
Shared data	A piece of data have to hold exactly the same value among the different artifacts involved.
Data loading	A piece of data from one of the file involved is loaded by the code in another file involved.
Generation	From one of the file involved the other files involved are completely or partially generated. Also the modification of part of a file is accepted.
Description	One of the file involved contained a description of the content of another file (a part or the whole file).
Execution	One file execute the code contained in another file.

5.4.1 Shared ID - Example

A configuration file written in XML (Listing 5.1) contains the qualified name of a Java class (Listing 5.2). The class is named `S3FileSystem` and it is contained in package `org.apache.hadoop.fs.s3`; the fully qualified name is therefore `org.apache.hadoop.fs.s3.S3FileSystem`.

Listing 5.1. Snippet from file File src/java/core-default.xml at commit 1058343

```
<property>
  <name>fs.s3.impl</name>
  <value>org.apache.hadoop.fs.s3.S3FileSystem</value>
  <description>The FileSystem for s3: uris.</description>
</property>
```

Listing 5.2. Snippet from file File src/java/core-default.xml at commit 1058343

```
public class S3FileSystem extends FileSystem {
```

5.4.2 Shared data - Example

Two different configuration files of Ivy have to specify the same version of a particular library. The library is the Google Protobuffer and the value of the version is "2.4.0a". The files involved are an XML file (Listing 5.3) and a properties file (Listing 5.4).

Listing 5.3. Snippet from file File ivy/hadoop-common-template.xml at commit 1134857

```
<dependency>
        <groupId>com.google.protobuf</groupId>
        <artifactId>protobuf-java</artifactId>
        <version>2.4.0a</version>
</dependency>
```

Listing 5.4. Snippet from file File ivy/libraries.properties at commit 1134857

```
protobuf.version=2.4.0a
```

5.4.3 Data loading - Example

Configuration data is loaded by Java code from an XML file, to implement a unit test on the `Configuration` class.

In Listing 5.5 you can read the code of the XML file, while in Listing 5.6 is reported the line responsible for loading the XML file.

Listing 5.5. Snippet from file File src/test/test-fake-default at commit 1126719

```
<!-- This file is a fake version of a "default" file like
  core-default or mapred-default, used for some of the unit tests.
  -->
<configuration>
  <property>
    <name>tests.fake-default.new-key</name>
    <value>tests.fake-default.value</value>
    <description>a default value for the "new"
         key of a deprecated pair.</description>
  </property>
</configuration>
```

Listing 5.6. Snippet from file src/test/core/org/apache/hadoop/conf/Test/ConfigurationDeprecation.java at commit 1126719

```
static {
        Configuration.addDefaultResource("test-fake-default.xml");
}
```

5.4.4 Generation - Example

A script used for setup may generate different files. For example the bash in Listing 5.7 file generates the actual mapred-site.xml from a template.

In this case the repository contains the template file but not the generated file, which would be present in a project using Hadoop.

Listing 5.7. Snippet from file src/main/packages/hadoop-setup-conf.sh at commit 1190035

```
...
template_generator ${HADOOP_PREFIX}/share/hadoop/common/
    templates/conf/mapred-site.xml ${HADOOP_CONF_DIR}/
    mapred-site.xml
...
```

5.4.5 Description - Example

The documentation of the Access Control List functionalities reported in Listing 5.8 describes a functionality expressed in class AccessControlList (path src/java/org/apache/hadoop/security/authorize/AccessControlList.java).

Listing 5.8. Snippet from file src/docs/src/documentation/content/xdocs/cluster_setup.xml at commit 998001

```
<tr>
        <td>mapreduce.cluster.acls.enabled</td>
        <td>Boolean, specifying whether checks for queue
            ACLs and job ACLs are to be done for authorizing
            users for doing queue operations and job
            operations.</td>
        <td>If <em>true</em>, queue ACLs are checked while
            submitting and administering jobs and job
            ACLs [..]. </td>
</tr>
```

5.4.6 Execution - Example

A POM file executes the code of Java class (see Listing 5.9).

Listing 5.9. Snippet from file pom.xml at commit 1195817

```
<doclet>org.apache.hadoop.classification.tools.
    IncludePublicAnnotationsStandardDoclet
</doclet>
```

5.5 Classification

The real project we selected for the analysis is Hadoop[1] (See step 1 in Section 7.4). We considered 39 bug-fixing commits from the Hadoop project (step 2), that were classified in [Vetro' et al., 2012] as cross-language (because they contain logical interactions). After a first inspection we discarded 3 commits because they were not cross-language (step 3).

Out of the remaining 36 commits we found semantic cross-language relations which we could classify in 27 cases (75%). More in details, in 11 commits we found one interaction, in 10 cases two relations, in 3 commits we found 3 relations, in two cases we found 4 occurrences, and in one case even 8 interactions.

We can conclude that using logical interaction as a proxy to identify semantic interactions has an estimated average precision of 75% with a 95% confidence interval ranging from 57% to 87%, estimated using a proportions test.

Figure 5.1 reports the frequency of the interaction categories. Of course here we report only the relations which we were able to identify. We cannot exclude the presence of other relations that we were unable to detect. Thus the number of *cross-language* interactions we identified could be interpreted as a lower-bound of possible existing relations. Some relations could be expressed implicitly, for example a file could load the content of another file using a library method of which we do not know the semantics.

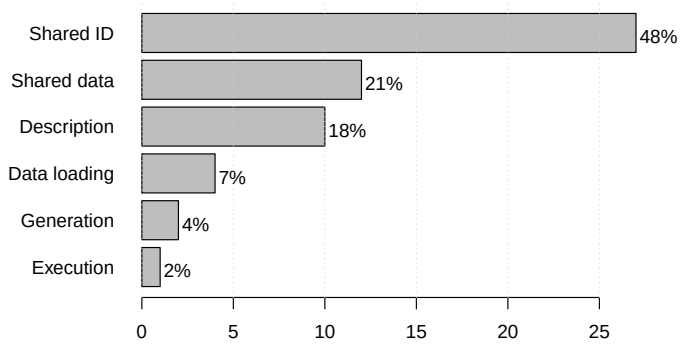

Figure 5.1. Frequency of semantic cross-language interaction categories

The most frequent category of relation is by far *Shared ID* (27 instances). In 12 cases we found a *Shared data* relation, in 10 *Description*, in 4 *Data loading*, in 2 *Generation*, and in 1 *Execution*. In this case of a *Generation* relation the repository normally contains the file which represents the source of the generation – typically

[1]http://hadoop.apache.org

a template file – but not the generated file, which would be present in a running configuration of the system.

The most frequently involved files were xml (42 cases), followed by java (30), properties (16) and sh (11). In 3 cases each also ac, am and spec files were involved. In only one case we found file with avpr and c extension.

5.6 Summary

We conducted and investigation of the cross-language semantic interactions in an open-source project. A very simple approach based on logic links – co-presence in the same commit – is able to indicate the presence of confirmed semantic interactions with a limited though acceptable precision (75%).

Based on the actual instances we defined a taxonomy of semantic interactions, which provide us with a deeper understand of cross-language relations. The relations we individuated are: *Data loading*, *Description*, *Execution*, *Generation*, *Shared data*, *Shared ID*.

We also computed the frequency of occurrence of the individual categories. Apparently about 50% of the interactions take places by means of shared ids.

An ongoing work is being devoted to the implementation of tool support for cross-language interactions. The knowledge on the interaction categories is the main starting point for designing tool support for cross-language interactions. In addition the information about the frequency allow defining priorities among the different interaction categories when building the supporting tool.

Chapter 6

A preliminary empirical assessment on the effects of cross-language interactions

In this Chapter we look into the effects of cross-language interactions, as needed to answer the thesis research question **RQ B.2**. This Chapter is based on a previous work published as [Vetro' et al., 2012]. That work was realized jointly with Antonio Vetrò, Marco Torchiano, and Maurizio Morisio.

6.1 Definitions

Before stating our goals and translating them into actionable research questions, we define how we do identify and measure the languages interaction. We provide here a list of definitions used throughout the rest of the paper.

Definition 1. Module: we considered a module each single file.

We consider a commit[1] as a unit of work, consequently we suppose that files committed together are related.

Definition 2. Intra-language commit (ILC): a commit containing a set of modules with the same extension.

Definition 3. Cross-language commit (CLC): a commit containing modules with different extensions.

[1] We refer to the term commit as used in the context of version control systems.

Definition 4. Cross-language commit for an extension (CLC_{ext}): a CLC containing that includes modules with the extension ext.

Definition 5. Defect fix: a commit executed to fix a defect.

We consider a module to be cross language when it is related to modules written in a different language (e.g., a Java file loading the configuration from an XML file). To measure how much a module is cross language we analyze its history: if the module was frequently committed with files written in other languages we consider that as an indicator of interaction between the module and those files. This interaction is measured through different variants of the cross language ratio (CLR).

Definition 6. Cross language ratio of a module (CLR_m): the CLR of a module m is the fraction of cross-language commits in which m was involved with regard to the total number of commits regarding the module (both intra-language and cross-language).

Definition 7. Cross language ratio of a module with regard to an extension ($CLR_{m,ext}$): the CLR of a module m considering as CLC only the commits involving m and a module with extension ext.

Definition 8. Cross language ratio of an extension (CLR_{ext}): for each extension ext we compute its cross language ratio as the mean of the CLR_m considering all modules having extension ext.

Definition 9. Cross language ratio of an extension $extA$ with respect to an extension $extB$ ($CLR_{extA,extB}$): the mean of $CLR_{m,extB}$ among all modules m with extension $extA$.

Definition 10. Cross Language Module (CLM): a module is cross language if its CLR is $\geq t_{CLM}\%$, where t_{CLM} is a threshold to be defined.

Definition 11. Intra Language Modules (ILM): a module is intra language if its CLR is $< t_{ILM}\%$, where t_{ILM} is a threshold to be defined.

6.2 Design

The goal of this preliminary study is two-fold. Firstly we investigate the level of languages interaction in a common project. Secondly, we verify whether the level of interaction is related to quality problems. We look at defects as a proxy of software external quality. We identify two research questions related to the first goal.

RQ1 *How much interaction is there among the languages present in a project?*

The interaction is computed as the percentage of CLC among a set of commits. First we consider all type of commits (RQ1.1), then (RQ1.2) we consider separately the commits related to a particular activity (e.g., improvement, bug fixing, new feature).

Once we have defined the size of the phenomenon by answering to RQ1, we will go deeper considering the behavior of each single extension.

RQ2 *Which extensions interact more?*

The second research question is answered at two levels, i.e. firstly investigating the relationship between one extension versus all the other extensions (RQ2.1), then analyzing the most interacting pairs of extensions (RQ2.2).

We answer RQ2.1 computing the for each extension, while we answer RQ 2.2 computing the for all pairs of extensions.

The last research question is related to the second goal, i.e. investigating whether a high interaction between languages might result in higher defect proneness.

RQ3 *Are Cross Language Modules more defect-prone?*

We answer RQ 3 computing the number of Cross Language Modules (CLM) with and without defects, and the number of Intra Language Modules (ILM) also with and without defects. Then we compare the two proportions with/without defects by means of the F-test to see whether the proportion of Cross Language Modules with defects is different from the one of Intra Language modules.

This metric is computed at three granularity levels:

- considering all files regardless of their extension (RQ3.1),
- considering for each single extension its level of interaction with all the other extensions as aggregate (RQ3.2),
- considering interaction between specific ordered pairs of extensions (RQ3.3).

6.3 Case study

This exploratory study aims at understanding the phenomenon of language interaction and derived quality issues. We also use it to investigate whether the methodology defined above is applicable. We selected as a case study Apache Hadoop[2] , which is a set of libraries to support distributed data processing. We selected

[2] http://hadoop.apache.org

Hadoop because it is a mature project (it is supported since April 2006) and it is used in many industrial applications (e.g., Yahoo, and Facebook).

Our methodology for computing the metrics defined above is based upon the fact that Hadoop uses SVN[3] to manage artifacts versions and JIRA[4] to track not only defects but any other activity that can be associated with software artifacts. Those elements are called *JIRA issues*, and each project has its own set of issues. Example of JIRA issues are the implementation of a new feature, a single implementation task, a bug report, and so on. Hadoop developers established links between commits in the SVN code repository to JIRA issues by systematically including issue ids in their SVN commit comments.

We downloaded the SVN log from the Hadoop repository (last revision retrieved is the 1233090, from 01/18/2012, the first available revision is the 776174 from 5/19/2009). We also extracted all JIRA issues from the Apache JIRA database.

We computed all modules CLRm and observed their distribution: about 30% of modules have CLR_m between 0 and 0.1, and about 55% files have CLR_m between 0.9 and 1. Given these percentage and given that the remaining files have a positive (right) skewed distribution, we decided to use as thresholds t_{CLM}=t_{ILM}=50% to define CLM and ILM modules.

All	Bug	Improvement	NewFeature	Subtask	Task	Test
0.53	0.12	0.26	0.30	0.45	0.26	0.05

Table 6.1. Percentage of cross language commits (RQ 1)

CLRext	Nr files	Extension
0.96	49	c
0.87	114	sh
0.72	75	properties
0.71	320	xml
0.59	4328	java

Table 6.2. CLR_{ext} (RQ 2.1)

[3] `http://subversion.tigris.org/`

[4] `http://www.atlassian.com/software/jira/overview`

extA/extB	C	Java	Properties	Sh	XML
C	-	0.51	0.10	0.50	0.83
Java	0.01	-	0.28	0.04	0.48
Properties	0	0.54	-	0.36	0.46
Sh	0.09	0.22	0.24	-	0.47
Xml	0.04	0.52	0.43	0.24	-

Table 6.3. $CLR_{extA,extB}$ (RQ 2.2)

	C	Java	Properties	sh	XML
C	-	Inf	0	0	Inf
Java	2.79	-	0.32	0.43	0.96
Properties	Inf	1	-	12.08	0.94
Sh	3.55	4.45	17.17	-	7.44
Xml	3.83	0.95	3.22	4.73	-

Table 6.4. Odds ratio of the defectivity in respect to the relation between pairs of extensions (RQ 3.3)

6.4 Results and discussion

Table 6.1 reports the percentage of cross language commits in the Hadoop repository: 53% of all commits (first column) are CLC, i.e. containing files of different languages. Looking at the portion of CLC related to the different activities (i.e., JIRA issues), we observe that their percentage varies with respect to the type of issue (from 2nd to last column in Table 6.1). It goes from a minimum of 5% in commits related to Test up to a maximum of 45% in Sub Tasks (since not all issues are linked to JIRA issues, the mean "All" in the first column is not related to the other means in the following columns).

RQ 1.1 answer the 53% of commits in Hadoop are cross language.

RQ 1.2 answer looking at the single activities, we derive that writing/modifying tests or fixing bugs are activities that involve mainly a single language, while adding new features is an activity that involves multiple types (or at least extensions).

We now proceed to RQ 2.1 and 2.2. Table 6.2 contains the top 5 extensions in terms of number of files: c, sh, properties, xml and java. Among them, four extensions correspond to programming languages and one is used for configuration files. Subsequently, we compute the CLRextA,extB for all combinations of the five extensions . Table 6.3 reports the CLRextA,extB.

RQ 2.1 answer all most common extensions in Hadoop are highly interacting with other extensions (i.e., $CLR_{ext} > 0.50$).

RQ 2.2 answer the most frequent interactions ($CLR_{extA,extB} \geq 0.50$) are: C-XML (0.83), Properties-Java (0.54), XML-Java (0.52), C-Java (0.51), C-sh(0.50). Border values are: Java-XML (0.48), sh-XML (0.47) Properties-XML (0.46), and XML-Properties (0.43).

We observe that the only pairs with frequent interactions in both directions are Java-XML and Properties-XML. All the other pairs have frequent interactions in only one direction. For instance, $CLR_{XML,C} = 0.04$ and $CLR_{C,XML}$=0.83 means that most of the commits involving C contain also XML files, but not the other way around.

We now focus on the last RQ, i.e. on the relation between languages interaction and defect proneness. Table 6.5 contains metrics to answer RQ 3.1 (first line) and RQ 3.2 (from 2nd to last line). The following columns contain, in the order: the number of ILM with no defects and then with at least one defect, the number of CLM with no defects and then with at least one defect, the p-value of the F-test and finally the odds ratios (which is greater than 1 when CLM are more defect prone than ILM).

RQ 3.1 answer considering all extensions, ILM are more defect prone that CLM (about 5 times less).

RQ 3.2 answer considering the five most common extensions, we observe that three extensions (XML, Properties and C) have CLM with higher defect proneness, while two extensions (Java and Sh) exhibit the opposite relation.

Among the above differences, only all extensions and Java are statistically significant (p-value ≤ 0.05).

Finally, Table 6.4 contains the odds for each pair of extensions to answer to RQ 3.3. We report in bold the values for which we obtained a p-value ≤ 0.05. We observe 7 pairs for which ILM are less defect prone than CLM, 12 pairs with CLM more defect prone than ILM and one pair with odds ratio =1. We consider only values with p-value ≤ 0.05 to answer RQ 3.3.

RQ 3.3 answer four extension pairs have CLM more defect prone then ILM (C-Java, C-XML, Properties-C, Sh-C),

five extension pairs have ILM more defect prone then CLM (C-Properties, C-sh, Java-XML, Properties-XML, XML-Java)

one extension pair have exactly same defect proneness (Properties-Java).

We notice that interactions where CLM results more defect prone involve always the C files. While interactions where ILM results more defect prone involve mainly XML, however C is also present. An interesting fact is that the pair Sh-C is in the first set, the pair C-sh is in the second.

	RQ	MN	MY	CN	CY	P	Odds
all	2	1891	225	2875	89	0.000	0.26
c	2.1	2	0	46	1	1.000	Inf
java	2.1	1692	201	2239	25	0.000	0.09
properties	2.1	19	1	45	7	0.429	2.92
sh	2.1	10	5	64	13	0.162	0.41
xml	2.1	96	11	184	24	0.851	1.14

Table 6.5. Relation between classification in ILM and CLM and presence of defects (RQ 3.1 and 3.2)

Besides these considerations, we do not have an unique answer for RQ 3. However, we observe that having languages interacting with other languages is related to higher defect proneness for certain languages (mainly C) and specific interactions.

6.5 Threats to validity

Internal: in this exploratory case-study different aspects were not considered. In particular we did not examine all the possible confounding factors influencing the defect proneness of the modules. Among them the age and the size of modules (expressed in LOC, for example) are the most relevant ones.

We discriminated between modules on their names while the same module can change name in the course of the project. We grouped the files by their extension while a different extension could not always indicate a different language.

Construction: we are unable to measure directly the interaction between modules written in different languages and consequently we use as a proxy their concurrent presence in the same commits, which may be an imprecise approximation.

External: another threat is due to selection bias: we have no particular reason to believe that Hadoop is representative of other software projects. Of course having considered only one project generalization of the results presented is not possible at all.

6.6 Summary

Although we do not have unique answers, the results and observations from this exploratory study let us understand that the problem is worthy to be investigated. In fact we observed that more than half of the commits in Hadoop are cross language (at least according to our definition). However we also observed that this property depends on the type of the activities and the extensions of the modules.

Commits related to testing or fixing bugs involve mainly a single language, while adding new features or doing implementation sub-task are activities which involve multiple languages (or at least extensions).

Looking at the single extensions, we verified that the most common extensions are frequently changed together with files with different extensions. Frequent interactions are generally not symmetric, and many of them involve XML.

When we look at defect proneness, we observe that for Java modules the interactions with other languages (as an aggregate) is not problematic at all: we observed that Java CLMs files are ten times less defect prone than ILMs. However, when looking at single pairs of interactions, we notice that several pairs have CLM significantly more defect prone then ILM, especially C modules. Finally, the widespread interaction between Java and XML apparently is not related to defect proneness.

Chapter 7

Spotting automatically cross language interactions

In this Chapter we present a method for automatically spotting cross-language interactions, to answer the thesis research question **RQ B.3**. This work was previously published as [Tomassetti et al., 2014]. It was a joint work realized with Giuseppe Rizzo, and Marco Torchiano.

7.1 Introduction

Most of the applications realized today are composed by artifacts written in different languages. The Web offers an excellent case study, since the majority of the applications use several languages for both server and client side. On the server side the typical scenario includes at least a general purpose language (GPL), SQL and some formats to store configuration (XML, JSON, etc.). On the client side HTML, CSS and Javascript are typically adopted. The different artifacts cooperate to execute some tasks, as part of the application, therefore they have to communicate and be linked together: a certain CSS rule affects a given tag, a XML file describes which Java classes have to be instantiated, the execution of a Ruby script is affected by the configuration reported in a YAML[1] file. Different languages can be mixed even in the same artifact, for instance consider CSS or Javascript in HTML pages or a Java function call receiving a string which happens to be SQL code. Framework or single projects can also adopt Domain Specific Languages (DSL) to realize specific facets of the complex system. In Listing 7.1 we report an example of a Java statement specifying a query to a database using the Hibernate Query Language (HQL), a DSL resembling SQL. The query retrieves all the rows from the table `Employee`. The

[1] `http://yaml.org`

resulting rows are then converted into corresponding Java objects by Hibernate[2], a well-known Object-Relational Mapper. By convention tables and corresponding Java classes have the same name (`Employee` in this case), therefore there is a cross-language relation between the Java class and the table reference in the query: if one of them changes, developers should consider to update the others. The specific rules which determine how the artifacts are composed depend on the language and the framework used: each can use its own logic to operate, creating run-time relations between artifacts. Considering Listing 7.1, the fact this particular call receives a string supposed to be a valid HQL code is defined by the implementation of Hibernate. Instead conventions are usually not formalized explicitly, but are nevertheless relevant to favor communication between developers and ease comprehension.

Listing 7.1. A snippet of HQL code in a Java statement.

```
List<Employee> employees = session.createQuery(
    "FROM Employee").list();
```

The rules for cross-language relations, being determined by framework implementations or by conventions, have to be studied and to be always considered during the development. A violation of a hard rule leads to errors which are difficult to track because implicit relations spread across different files in different languages have to be considered all together. Hence, even just renaming a Java class can lead to a runtime error because the name of the class was not updated in few XML and property files referring to it. Violating conventions could instead lead to a code which is harder to maintain, because developers rely on these conventions for comprehension. While modern IDEs offer support to identify inconsistencies between two files written in the same language, the developer is typically left on his own when it comes to cross-language relations. Without refactoring support the developer has to replicate manually the update in all related artifacts. Without navigation support cross-language references are not immediately apparent, the developer has to know and remember the cross-language rules determining relations and to manually navigate to other files for retrieving related information. Without validation support, a broken link is not immediately apparent. An experiment performed by Pfeiffer *et al.* [Pfeiffer and Wasowski, 2012a] shows that tool support for cross-language relation can greatly improve the developer performance. However all the existing approaches are framework specific: they require to manually specify the rules which govern the relations expected by a certain library. Each new language, each new DSL, each new framework require to adapt these tools. Considering that formalizing cross-language relations precisely can be quite difficult per se [Mayer and Schroeder, 2013] this leads to a considerable effort to implement and maintain cross-language

[2] `http://hibernate.org`

relations support.

Taking inspiration by this, in this paper we motivate the following research question: is it possible to automatically spot cross-language relations in a variety of projects written with different languages? Exploiting the semantics of the language and relying on a predictive model, our approach is able to spot cross-language relations with 92.2% of F1[3]. The experiments have been conducted on an in-house benchmark which is, together with the source code of the framework and the experiment settings, publicly available at `http://github.com/CrossLanguageProject`. The reminder of the paper is organized as follows. In Section 7.2 we further explain our motivation and prior work. We present the benchmark used for the experiments in Section 7.3. Our approach is presented in Section 7.4 and in Section 7.5 we present the experiment results. Finally in Section 7.6 we discuss the results and we foresee possible outlook.

7.2 Related work

Research attempts on cross-language relations are quite recent. Generally they can be summarized as: *i)* to offer a classification of cross-language relations, *ii)* to characterize empirically the effects of cross-language relations, *iii)* to provide prototyping tool support.

Classification: in [Tomassetti et al., 2013b] the authors presented a classification of different forms of cross-language relations, identifying six different types: shared ID, shared data, data loading, generation, description, and execution. Of all those types the most commonly used is shared ID. In the context of this work we focus exclusively on this kind of cross-language relations. Meyer and Schroeder [Mayer and Schroeder, 2013] classify exclusively cross-language links implemented in Java framework in respect to XML artifacts. The relations they consider correspond to the category "shared ID", according to the classification proposed in [Tomassetti et al., 2013b]. They built metamodels of the languages involved (Java and XML) for this particular purpose and specified the rules controlling the cross-language relations of three Java framework, deriving from them common patterns. Among the main results, they report that specifying manually rules for cross-languages relations is difficult.

Empirical results: in [Vetro' et al., 2012] the authors investigated how many of the commits of the Hadoop[4] project involved more than one language and the

[3]By F1 we mean the F-Measure with $\beta = 1$. It corresponds to the harmonic mean of precision and recall.

[4]`http://hadoop.apache.org`

effect of being involved in cross-language commits on defectivity. Results show that some relations are particularly negative. However results of this paper are based on a coarse proxy for the identification of cross-language relations; considering projects hosted on a repository, they relied on the logs for looking at the files which have been committed at the same time. But this approach leaves the burden of spotting manually the cross-language relations. In fact, a method to automatically identify cross-language relations at a finer level can permit more precise empirical investigation on their effects on a large scale, where manual identification is not feasible. Pfeiffer *et al.* [Pfeiffer and Wasowski, 2012a] used TexMo in a controlled experiments with 22 subjects to demonstrate the effects of tool support for cross-language references. According to their results, developers having access to tool support for cross-language references were significantly faster and more frequently correct in locating sources of errors. Developers without this type of support instead have difficulty to reconduct the errors which they encountered at run-time to their ultimate source, a broken cross-language relation.

Specific tool support: possible solution to the problem consist in *i)* developing specific IDE support, *ii)* substituting existing languages with families of integrated languages, *iii)* implementing proper language integration inside language workbenches. The first approach was adopted by Pfeiffer *et al.* [Pfeiffer and Wasowski, 2012b], [Pfeiffer and Wasowski, 2013]: they realized different prototypes integration tool support for cross-language relations into mainstream IDEs named as TexMo and Tengi. An example of *family of languages* comes from Groenewegen *et al.* [Groenewegen and Visser, 2008]: upon observing that the amalgam of languages used in a single web application project are typically poorly integrated they proposed the adoption of an unique language to model all the different concerns of web applications: WebDSL. Finally regarding *language integration* in the context of Language Workbenches is described by Tolvanen *et al.* [Tolvanen and Kelly, 2010]. In their paper they describe their experience in integrating Domain Specific Modeling (DSM) languages. They considered only DSM realized in the context of the MetaEdit+ system, without integration with GPLs. GPLs integration is instead possible in another Language Workbench: Jetbrains MPS. An example in this direction is described in this paper [Tomassetti et al., 2013c]. Integration in mainstream IDEs has the great advantage to leverage environments which are already familiar to most of the developers, but they require the implementation of specific support for each single framework considered. On the contrary, the other solutions require a migration but offer deeper integration, attainable with a limited effort.

7.3 Benchmark

To the best of our knowledge, no other research attempts have been spent to spot at fine grained level the cross-language relations. It results in a lack of gold standards for benchmarking the performance of proposed approaches. To fill this gap, in this paper we propose an in-house benchmark as compendium of our approach. As described previously, the Web offers a vast number of projects written using different formal languages each. In addition, the most used formal languages for Web applications (HTML, JS, CSS) share intrinsically numerous cross-language relations, which are usually hidden, making the task extremely challenging. We have then selected a web project based on the AngularJS[5] framework named angular-puzzle[6]. Table 7.1 summarizes the statistics of the proposed benchmark.

Description	Values
no. of different files	12
no. of formal languages involved	4
no. of lines among all files	2927
no. of manually identified cross-language relations	142

Table 7.1. Statistics of the benchmark proposed in this paper. Any artifact is considered, excluded pictures. The number of language involved considers also the natural language text. The number of cross-language relations is computed considering HTML and JS artifacts (excluded lib artifacts).

Two human experts have been involved in the creation of the benchmark; the aim was to manually detect the cross-language relations between artifacts of two different extensions (JS and HTML), to which we excluded the AngularJS library artifacts. We also excluded the CSS files, since the identification of cross-language relations is easier to be spotted due to the tag selection. The relations have been reported pair-wise; the dataset lists the *src* and *dst* files, *row* and *column* where the relation have been spotted from both artifacts and the *surface form* (shared ID) of the relation per each cross-language relation. The overall agreement score reached so far was good. After the first annotation step, we have dedicated a cleansing step for fixing the errata spots. The benchmark is released as public license at `http://github.com/CrossLanguageProject/goldstandards`.

[5] `http://angularjs.org`

[6] `http://github.com/pdanis/angular-puzzle`

7.4 Method

Most of the cross-language relations are implemented using a shared identifier. For example a Javascript statement could refer a particular tag in a HTML document by its ID. When one of the two ends of the relation is changed, the other one has also to be updated, if the relation wants to be preserved. However not all the instances of the same terms are related, because not all of them identify the same entity. For instance, if we consider pairs of instances of the same term appearing in different files, written in different languages, chances are high that the two identified entities will be different and unrelated. Our method aims to automatically and independently identify the cross-language relations from the languages considered and the used framework.

To perform the classification we factorize the AST using the candidate spots as pivots and we exploit the *context* of each pair. The semantic similarity measure between each pair of local factorized ASTs is computed. In details the proposed method consists of the following steps:

- for each artifact an AST is derived. An AST may host sub-ASTs of different languages, corresponding to snippets of embedded languages;
- for each node the set of nodes corresponding to its context is collected;
- pairs of nodes corresponding to the same term and enclosed in artifacts of different formal languages are spotted as potential candidates. For each pair a semantic similarity is computed. This process is meant to narrow down a set of features which are then used by the classification stage;
- all the candidate spots become observations of the classification model; of these pairs of nodes the ones related will be positive examples, the candidate which are not related but hold the same term will be the negative examples. The learning stage is used to train the algorithm of the predictive model.

7.4.1 ASTs construction

The first stage of the process is to map each source file to its corresponding AST representation. This allows to distinguish between keywords and relevant values present in the source code (i.e. identifiers and literals) and to organize the data in a logical structure on which is possible to reason about relations between nodes. The host language of a file is determined by its extension (the assumption made is grounded on the fact that a Java file usually contains a host Java AST). Inside the host AST, foreign ASTs can be added. They represent snippets of other languages embedded in the original file.

Consider the example shown in Figure 7.1: a snippet of HTML is reported. The attribute `onclick` of the `div` tag contains Javascript code, in particular a call to the function `showStats`. From this piece of code is derived an AST having as root a HTML node. The Javascript snippet is appropriately parsed and a corresponding Javascript AST is obtained. The Javascript AST is then embedded in the host HTML AST as a child of the HTML attribute which contains the Javascript code.

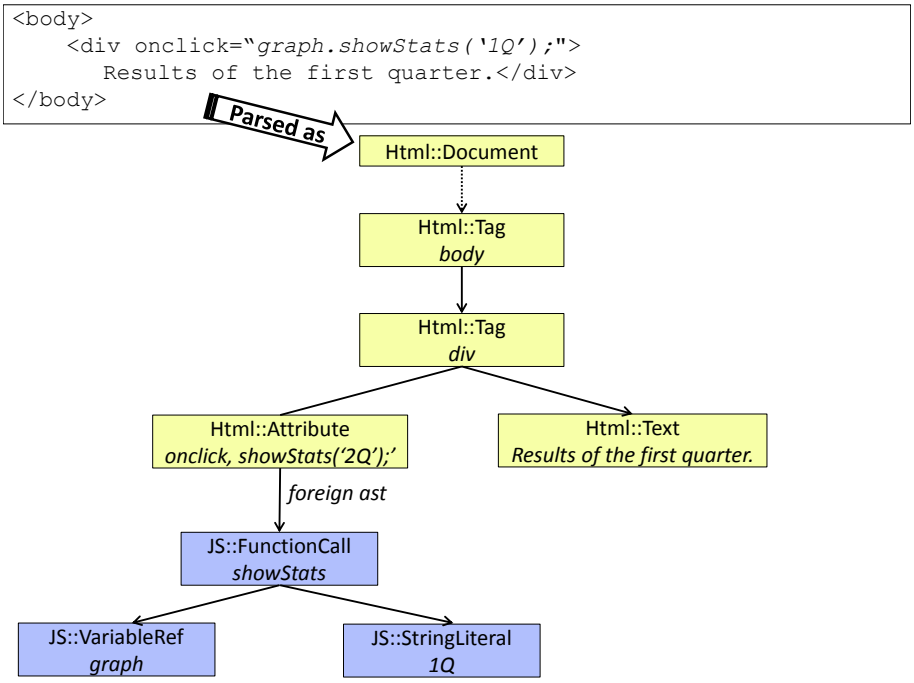

Figure 7.1. A Javascript AST embedded in a HTML AST.

7.4.2 Context

The role of the context for spotting cross-language relations has been previously introduced by Mayer *et al.* [Mayer and Schroeder, 2013]. Inspired by this preliminary consideration, we have started over an exhaustive investigation on the context in the field of spotting automatically cross-language relations. We then consider important the context to discriminate between instances of terms that just happen to have the same surface form from instances which are concretely related.

Consider the example shown in Figure 7.2: the term `title` appears two times in *index.html* and five times in *app.js*. The first appearance in *index.html* is related to the first two appearances in *app.js* while the remaining instances in the two files are also reciprocally related. They can be intuitively distinguished on the basis that

the first group of instances is hosted in the context of `types`, while the second is hosted in the context of `puzzles`.

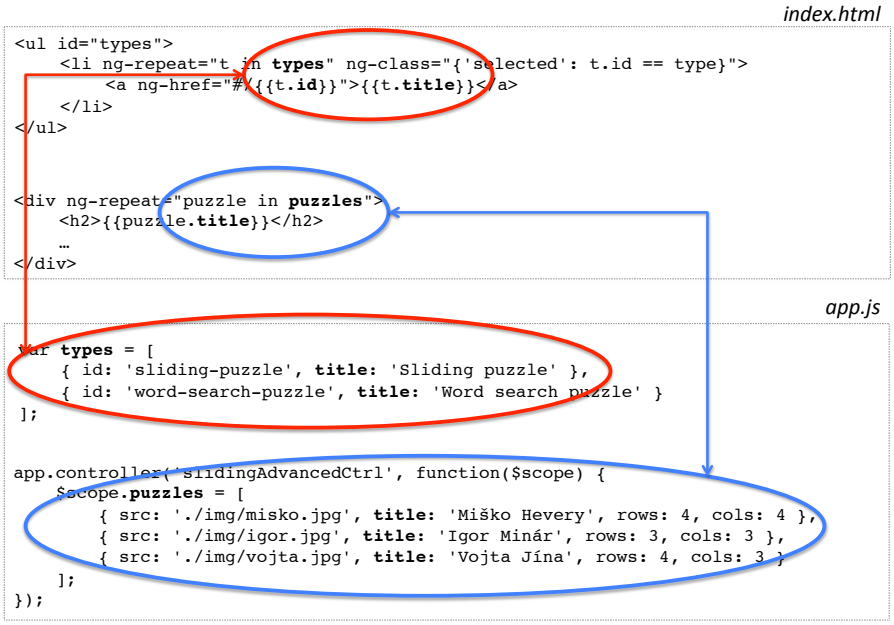

Figure 7.2. Example of cross language relations organized in hierarchies.

To translate this consideration into a formalized approach we devise an algorithm to traverse siblings node in ASTs which is language agnostic. More details about the context extraction algorithms can be found in the source code of the project. The outcome of the context identification steps is a set of AST nodes which constitute the context of a given input node.

7.4.3 Features derivation

The candidate pairs of nodes, which share the same surface forms, are compared by means of their contexts; the resulting comparison is therefore executed on the two contexts. From each pair of contexts two different set of surface forms are extracted. These sets are compared using state-of-the-art algorithms for instance matching such as Levenshtein algorithm (working at word grained level), Jaccard, Jaro, and Tversky [Navarro, 2001]. These algorithms provide a coefficient of similarity, when the coefficient is superior to a given threshold the nodes are considered related, otherwise they are not. Therefore they provide a raw idea whether a candidate spot

actually defines a valid cross-language relation. For this matter, we have considered them as baselines in our proposal.

Further features have been derived from the context similarity, such as: the number of words appearing in both contexts; the sum of tf-idf value of the words appearing in both contexts; the sum of the itf-idf value of the words appearing in both context (the itf is defined as $log(1/tf)$); the percentage of words of each of the two contexts which appear also in the other; the number of words of each of the two contexts which do not appear also in the other; the percentage of words of each of the two contexts which do not appear also in the other. Table 7.2 proposes a recap of the above features.

7.4.4 Classification

Starting from the derived features, we map the task of spotting cross-language relations to a predictive task. The features are meant to: *i)* define the model and *ii)* train the classifier. Inspired by Mayer *et al.* [Mayer and Schroeder, 2013] who proposed a set of manually created rules, we built the classifier in order to create a list of rules that can automatically predict whether the observation is actually a cross-language relation. To achieve such a scope we used the Random Tree (RT) algorithm. We then compared this algorithm with K-nearest neighbors (K-NN) and Naive Bayes (NB) algorithms.

7.5 Experiment and results

The benchmark proposed above reports all the cross-language relations that two human experts have manually spotted in the context of the angular-puzzle project. Anyway, such a task covers only the true positive spots of the domain, leaving the burden to spot also the true negative ones. The population of interest includes all the pairs of nodes which i) are contained in files with different extensions, and ii) share a common word; the negative ones are those pairs which satisfy these conditions and are not semantically related yet. These latter pairs were automatically individuated. The union of the two sets forms the benchmark over which we have run our experiments. Hence, the results of our experiments are reported in terms of correctly spots of cross-language relation in case of actually a positive case (the pair holds a cross-relation) or correctly spot that the pair does not hold a valid cross-language relation.

Using a correlation matrix, we verify the correlation each feature has with the class to predict (originally the class spans from positive when the observation details a cross-language relation, negative otherwise). The results are reported in Table 7.3.

We performed a 10-fold cross validation and used WEKA-3.7.9[7] for running the classifiers. Table 7.4 reports the figures achieved so far by our approach according to three different classifiers. The RT performs better such kind of task, proving the intuition that Mayer *et al.* had in their paper, that a rule based approach can help for deciding whether a shared ID actually holds a cross-language relation. Our approach is extremely competitive, nearly solving the problem.

Finally, in Table 7.5 we compare the figures achieved by our approach with the ones obtained by simple approaches which leverage on instance matching algorithms. It is evident how the cross-language spotting task cannot be solved by just context similarity evaluation.

7.6 Discussion and outlook

We believe in the benefit of adopting the most suitable language to implement each facet of the system. Using the best language for the task leads to polyglot systems which include artifacts written in many languages; it also requires proper coordination to smooth the development, improving productivity and the quality of the developed systems. To obtain good language integration we need first of all to recognize cross-language interactions: the approach detailed in this paper aims to do that and it seems promising, considering the results obtained on the proposed case-study.

In this work we proposed a language agnostic approach to spot automatically cross-language relations. To measure the goodness of the approach we have created an in-house benchmark, which, to the best of our knowledge, is a first attempt in this direction. We believe that the case study we chose is particularly daunting for the presence of different mechanisms of interactions and the mix of languages appearing even in the same artifacts. Since we are looking forward to enlarge the current benchmark, we have released it publicly and we hope that the community will contribute to extend and improve it: refinements of our work as well as alternative solutions will be easily compared in this way. Using a predictive algorithm, we are able to nearly solve the problem of spotting cross-language relations with a F1 of 92.2%. The features used have been extracted from the context of the factorized ASTs compared pairwise. Alternative solutions are neither generic nor automatic, therefore they require a major framework-specific effort. On the other hand they offer perfect precision and recall, for the kind of cross-language relations they address, while a fully automatic and general approach cannot.

Our prototypal implementations is based on a library wrapping a set of existing

[7] `http://www.cs.waikato.ac.nz/ml/weka`

parsers[8]. Support for HTML, Javascript, Ruby, Java, XML, and properties files is already implemented. This implementation is offered to the community with the hope it will serve as a component for the realization of alternative approaches. Our implementation is designed in such a way that the support for other languages may be easily plugged in. To support additional languages all the language-specific work that is required is: *i)* thin wrapper around an existing parser, *ii)* the specification of the conditions under which another language can be added in the one considered.

As next step we want to work on generalization of our approach: we want to test the devised algorithms on different projects, realized using different frameworks. Then we plan to conduct empirical validation of *i)* the performance of our approach and *ii)* the benefits of such approach of polyglot software development. We also plan to further investigate cross-language relations which span to more than 2 different artifacts by time. That means considering the lattice of the combinations among the nodes extracted from the different artifacts.

[8]See `https://github.com/ftomassetti/codemodels` and its plugins.

Name	Description
shared length	number of words appearing in both contexts
shared tf-idf	sum of tf-idf value of the words appearing in both contexts. See `http://en.wikipedia.org/wiki/Tf-idf`
shared itf-idf	sum of the itf-idf value of the words appearing in both context. The itf is defined as $log(1/tf)$
perc. shared length [min,max]	the percentage of words of each of the two contexts which appear also in the other
diff [min,max]	the number of words of each of the two contexts which do not appear also in the other
perc. diff [min,max]	the percentage of words of each of the two contexts which do not appear also in the other
Levenshtein distance*	similarity distance between two set of chars (either two simple words or sequence of them). In this paper we use the second version of it
Jaccard distance	similarity distance between two sequences of words
Jaro distance	similarity distance between two sequences of words
Tversky distance	similarity distance between two sequences of words

Table 7.2. List of the features exploited in the proposed work.

	class
shared_length	0.0616
tfidf_shared	-0.0084
itfidf_shared	0.0861
perc_shared_length_min	0.0864
perc_shared_length_max	0.1703
diff_min	0.0560
diff_max	-0.0446
perc_diff_min	-0.0173
perc_diff_max	-0.0549
context	0.0374
jaccard	0.0969
jaro	0.0340
tversky	0.1061

Table 7.3. An excerpt of the correlation matrix, where we highlighted the correlation scores of the features to the class. From it we observe that tfidf, perc_diff_min, diff_max, and perc_diff_max are inversely correlated with the class to predict. For such a reason, we leave these four features out of the predictive model.

	P	R	F1
Naive Bayes (NB)	90.3	86.3	88.1
K-nearest neighbor (K-NN)	91.3	93.0	91.9
Random Tree (RT)	**91.6**	**93.2**	**92.2**

Table 7.4. Precision (p), Recall (r) and F-measure (F1) results of our approach using three different classifiers.

	P	R	F1
$\text{Levenstein}_{d=1}$	6.0	100	11.8
$\text{Tversky}_{d=0.8}$	12.9	43.8	19.9
$\text{Jaccard}_{d=0.8}$	13.9	35.0	19.9
Random Tree (RT)	**91.6**	**93.2**	**92.2**

Table 7.5. Precision (p), Recall (r) and F-measure (F1) results of the baselines and our proposed approach.

Chapter 8

Language integration using language workbenches

In this Chapter we present a prototype for language integration, to answer the thesis research question **RQ B.4**. This work was previously published as [Tomassetti et al., 2013c]. It was a joint work realized with Antonio Vetrò, Marco Torchiano, Markus Völter, and Bernd Kolb. Domenik Pavletic contributed to write Sect. 8.4.1.

8.1 Introduction

Multi-language systems development represents one of the crucial challenges in model software development [Mens et al., 2005]. In fact nowadays not only the size of software systems makes them complex, but also the large number of artifacts and the coexistence of distinct though interacting languages. As a matter of fact, the top 50 projects among the most active ones indexed by the *Ohloh* OSS directory[1] are composed, on average, by 16 distinct languages, ranging from a minimum of 3 (openSSH) to a maximum of 71 (Debian GNU/Linux).

The possibility of having different languages that interact and cooperate to deliver software functionalities adds flexibility and capabilities to software development. In fact the limitations of a language can be compensated with the capabilities offered by others. However, interaction of languages without proper integration and tool support might be a source for problems. As a matter of fact, we should consider that current tools typically check only the consistency within a set of artifacts written in the same language. For example, editors check that the methods invoked by an expression in Java code actually exist in the codebase, either in the same file or in another Java file. However, they are not able to control whether a piece of

[1] http://www.ohloh.net/

XML code used for configuration refers to a really existing Java class, because they are not aware of the cross-language semantics.

Problems due to language interactions have been in some cases addressed with ad-hoc solutions involving the development of specific supporting tools or plugins for different development platforms. This is the case, for instance, of the Spring tool suite which consists of a series of plugins for Eclipse offering support for the cross-references between Spring configuration files and Java code. Similar plugins are also available to handle references between Android XML configuration files and Java code.

However, in general verifying the consistency across the language boundaries is not possible because tools are not aware of the cross-language semantics.

These ad-hoc approaches are a symptom of the need for assuring some level of consistency of the global system, also across language boundaries. The major limit of those approaches is that they address the problem of supporting cross language references in an ad-hoc way, i.e. for a particular relation involving a specific pair of neighbor languages, which may result expensive and incomplete.

We advocate a mechanisms that offers tool support without involving the creation of specific editors for any peculiar use of a language, e.g. using XML for configuring a specific aspect of a given framework.

This paper first shows the relevance of these issues (Sect. 8.2), then provides some evidence concerning the problem deriving from language interactions (Sect. 8.3). After that it outlines a possible solution (Sect. 8.4) and describes related work (Sect. 8.5). Eventually research agenda is also provided to guide future works on this topic (Sect. 8.6).

8.2 Prevalence of Language Interactions

In previous Chapter 6 we presented an investigation on language interactions.We carried on a case study on the Hadoop project, to understand the magnitude of the phenomenon and identify possible implications. We started our investigation observing the commits in the version repository of Hadoop, driven by the following approximation: if a commit concerns files of different languages we assume that those files are related. For instance, considering a commit that fixes a bug and contains an HTML file and a CSS file: probably both files were changed in order to fix the bug.

We called cross language commits (CLC) those commits containing files of different languages. In particular we define *Cross Language Ratio* (CLR) as the ratio of CLC among all commits. In addition to the project CLR, we tried to understand which were the most interacting languages by measuring the CLR for each language, that is computed considering only the commits involving that language.

Table 8.1. Most interacting languages in a sample of five Apache projects

File Extension	Hadoop	Derby	Forrest	Harmony
All	53%	64%	66%	77%
C	96%	-	-	91%
sh	87%	92%	71%	74%
properties	72%	91%	77%	91%
XML	71%	88%	72%	94%
java	59%	62%	64%	77%
xsl	84%	100%	82%	99%
HTML	100%	87%	91%	99%

The rationale of computing this metric is to get a proxy of the level of interaction of files written in a given language. Assuming the bi-univocal correspondence language-extension, with this proxy it is possible to understand for each extension whether the majority of its files interact with files of other extensions.

In the Hadoop repository we observed CLR=53%. i.e. 53 out 100 commits in the repository were cross language. Further analyses (not yet published) on three more Apache projects (Derby, Forrest, Harmony) confirmed the initial observation: they show CLR respectively 64%, 66% and 77%.

Focusing on the specific languages we showed that the most interacting languages (among the ones with more files) of the Hadoop projects were C, sh, XML, Java and, considering also non programming languages, .properties files. CLR ranged from 59% (Java) to 96%(C) and even 100% (HTML, but with a lower number of files than C). Table 8.1 shows the CLR computed for common extensions in Hadoop and in three more projects, Derby, Forrest and Harmony: with the exclusion of C and sh, we observe very similar or higher CLR in the other projects. For Derby the most interacting extension is xsl, in Forrest is HTML while in Harmony both xsl and HTML.

These figures confirm the observation that the different languages used in software projects are not sealed off from each other, but they interact. In the next section we will discuss the problematic implications of such interactions, providing both anecdotal and empirical evidence.

8.3 Problems Given by Language Interactions

Combining different languages inside one system can lead to possible inconsistencies across the language boundaries.

In order to prove that interacting languages can be a source of problems in software projects, we are going to briefly provide and discuss examples of well-known cases (8.3.1), and to summarize the empirical evidence found in the previous work and in followup analyses (8.3.2).

8.3.1 Anectodal Evidence

In the literature, as well as in every developer's experience, there are several common examples of language interactions and related problems. Here we report a few specimens.

1) Web Applications: Web applications are developed using a plethora of languages. A typical web application uses a general purpose language for server-side processing, SQL to access the database, some template language (e.g., JSP or Facelets[2]) to generate the pages, HTML in the form of entire pages or snippets to be combined, Javascript for client-side elaborations, and CSS to control the appearance of the page.

All these languages cooperate to produce a working software system. References between artifacts written in different languages are therefore very common. Let us consider an HTML tag (e.g. `<input>`), with a specific *class* or an *id*. It can be referred to by:

- **Javascript:** for example, a function written in Javascript could verify the correctness of the input inserted by the user, if the tag is a field, or it can be used to react to some event generated by the user,
- **Server side language:** it could need to process the result of a submit including the value of that tag, if it is a field,
- **Template language:** it could need to generate javascript which refers to that particular id, or a css configuring that particular class,
- **CSS:** a CSS rule can be written to customize the appearance of a tag with that id or class.

A simple typing error in the name of the class or the id of the tag would be unnoticed by CSS (the rule will just not apply to the element), Javascript would tend to fail silently while it could cause a run-time failure on the server side.

[2]http://facelets.java.net/

2) XML Configuration of Java Applications:

Java applications rely frequently on configuration written in XML files, for example to configure Dependency Injection[3] or a Web Application framework[4].

These configuration files are used to achieve flexibility in the application. They commonly drive the instantiation and binding of classes, therefore they contain references to Java classes, expressed as strings corresponding to the name of the classes referred. The referred classes are expected to be present in the system, to extend a particular class, and to implement a given interface or possess some methods: for example a getter or setter for a particular property. When these conditions are not met the system can incur in a run-time error. There could be more complex constraints between the different elements referred, for example if a class is referred in some point, it should be initialized in another section of the file.

3) C and the Preprocessor:

Most of the files written in C (as well as files written in Objective-C or C++) host directives which are interpreted by the preprocessor and are expressed in its own peculiar language, which operates at the token level. The preprocessor directives constitute a language *per se* that could be used also in different contexts, i.e. outside C files. The interactions between these two languages – C and the language of the preprocessor – are source of different types of errors, which can be difficult to find and are detected only under particular configurations [Nie and Zhang, 2011]. In this case a language is embedded into the other. This is not the only case, consider for example the usage of SQL in different host languages.

For example a macro could obfuscate a C type or a C identifier, it could assume a value which makes the compilation fails or it can cause more subtle errors. Consider the example presented in Listing 8.1: here a function-like macro is used to calculate the square of a given number. It works as expected when it receives a number literal, but when it receives an expression having side-effects (like `a++`), it calculates the incorrect result.

Listing 8.1. An example of problematic language interaction between C and the preprocessor

```
#ifdef INLINE
#define square(x) ((x)*(x))
#else
#define square _square
#endif

int foo(int a){
```

[3]See for example Spring, http://www.springsource.org/ .

[4]See for example Apache Struts, http://struts.apache.org/ .

```
  return square(a++);
}
```

It is worth noting that while the preprocessor is frequently associated with C, it is completely unaware of the C semantics (except for the concept of comments and literals, which should be recognized to correctly identify preprocessor directives).

Typically the references across language boundaries are implemented by using a common identifier. Other possible ways of combining languages are described by Völter [Völter, 2011b] (at least for DSLs).

Unfortunately tools supporting any language are unaware of the type, the characteristics or even the existence of elements with that particular identifier among artifacts written in another language. Therefore the coherence of the whole system depends on human code inspections or verifications at run-time, when a failure, if noticed, can start an investigation of the problem.

This happens because the rules controlling language interactions are not explicitly formalized. Moreover typically modern IDEs support different languages through independent editors, which are hosted on a common platform (e.g., Eclipse). What is missing is a shared meta-model, permitting to express cross language concerns, and an environment supporting the expression of these concepts, and the enforcement of this constraints. Language workbenches offer that.

8.3.2 Empirical Evidence: Side Effects of Languages Interaction

In the previous preliminary work mentioned in Section 8.1, some of the authors of this paper have examined the role of language interactions on defect proneness. Starting from the definition of CLR (i.e., the ratio of cross language commits in all commits), they classified modules of Hadoop in Cross Language Modules (CLM), i.e. files with CLR $\geq 50\%$, and Intra Language Modules (ILM), i.e. files with CLR $< 50\%$.

Considering the five most interacting extensions in Hadoop, the authors observed that three extensions (XML, Properties and C) had CLM with statistically significant higher defect proneness, while two extensions (Java and sh) exhibit the opposite relation. Breaking down the analysis on specific pairs of extensions, we observed that:

- four extension pairs had CLM more defect prone then ILM (C-Java, C-XML, Properties-C, sh-C);
- five extension pairs had ILM more defect prone then CLM (C-Properties, C-sh, Java-XML, Properties-XML, XML-Java);

- one extension pair had exactly same defect proneness (Properties-Java).

Subsequent analyses on Forrest and Harmony projects revealed that Java (in Forrest), cpp and XML (in Harmony) were the languages whose Cross Language Modules were more defect prone than Intra Language Modules.

Although these observations do not provide univocal answers, they support the theory that particular interactions between languages could be problematic. We will discuss in the next section our proposed solution.

8.4 Language Integration in Language Workbenches

In Language Workbenches is possible to define different Language Components: we introduce the concept and present a classification in Sect. 8.4.1. Later we present our approach to integration in Language Workbenches (Sect. 8.4.2).

8.4.1 Language Components Classification

Language workbenches can be used to develop new programming or modeling languages, to extend the existing ones and mix different extensions potentially realized independently.

Language engineers can create modular *language components*. Each *language component* is a set of strictly related concepts used to implement a particular part of a language. A *language component* can implement either the core of a language or a single cohesive extension. A component can include different aspects of that particular part of a language. Typically language workbenches permit not only to implement proper aspects of the language and its notation (abstract and concrete syntax) but also the corresponding supporting tools (editors and code generators) which are necessary to make a language practically usable. A description of language aspects is reported in Subsection 8.4.1. A language component can depend on other language components. Developers can select a set of language components to be used in a particular project. The set of concepts defined by the components selected will determine which constructs can be used to realize the different artifacts included in the project.

Language components are the elementary blocks designed for reuse by language engineers and selected by developers. They represent an heterogeneous set of elements. To put some order in this large set we propose a classification based on five different dimensions: *independency*, *orthogonality*, *size*, *domain* and *completeness*.

For the **independency** dimension two possible values are allowed: *independent* or *extension*. An independent language component represents the core of a language

that can be used without any other language. Such a component could embed or reuse [Völter, 2013] some other language component (for example it could reuse a language component defining expressions) but the declared concepts can be used as root elements of artefacts, so they act as container of constructs of the language components included, not the opposite. An *extension* language component instead declares a set of concepts which could be used to complement concepts introduced by other components, permitting to improve the expressiveness of the original language. The concepts introduced can not be used to describe a model which makes sense per se.

The **orthogonality** defines how much a language component is reusable and in which contexts. It is expressed on a quality scale ranging from *very low* to *very high*. A language with an high orthogonality could be reused in every possible context while a language component with a very low orthogonality will introduce concepts to be reused for a very limited goal and only in particular contexts. Consider for example a language to represent requirements (high orthogonality) and a language to design acceptance tests for pension plans (low orthogonality).

The **size** of a language component can be calculated considering the number of concepts introduced, the number of properties and relations per concept. As a reference we report the dimensions of the language components considered .

The **domain** for which the language component was developed is another dimension. There are language components developed for a *technical* domain; for example programming languages. On the opposite there are *business oriented* domains. Consider for example a language to describe pension plans ().

Finally the **completeness** of a language component depends on the aspects described for that language component. While all the language components describe the abstract and concrete syntax of the concepts introduced, just a few specify the dataflow aspect.

Of course our proposed classification of languages is temptative. In some cases it is disputable the attribution to one particular value of one dimension. There are also connections between the dimensions: very often an *extension* (dimension *independency*) would have a lower orthogonality than an independent language.

Language aspects

A component can include different aspects of that particular part of a language. The set of language aspects which can be described depend on the nature of the language workbench used. Jetbrains MPS is quite comprehensive and permit to describe a large set of language aspects. Other language workbenches have frequently a smaller set of supported aspects. While abstract and concrete syntax have to be defined for each language component all the other aspects are optional.

This is a list of the language aspects we considered:

- **abstract syntax** it defines the underlying concepts expressable using the language, the relations between concepts and the information associated with each instance of a particular concept,
- **concrete syntax** it defines the notation, i.e. the way each concept is represented,
- **typesystem rules** this aspects collect rules for calculating the type associated with different contexts and conversion rules between types,
- **dataflow** it defines in which order and under which condition the code specified using the language is executed. For example it would specify that the code contained in the *then* block of an `if` is executed only when the condition is true while the condition itself is always evaluated,
- **translation** this aspect specifies how to translate from the language defined to a lower level language. It can specify different intermediate transformations until the code is translated to a language which can be compiled or interpreted,
- **constraints** it defines constraints for the relations between elements or semantic error checking (syntactic error checking can be directly obtained from the abstract and concrete syntax),
- **refactoring** this aspect collects refactoring operations which can be invoked from the editor on elements of the language. An example is the possibility to generate automatically getter and setter for a field.

Example of classification

Table 8.2. j.m stands for jetbrains.mps, c.m.c stands for com.mbeddr.core, c.m.cc stands for com.mbeddr.cc. ind. = independent, ext. = extension, sem. = semantic, syn = syntactic. Regarding the completeness the abstract and concrete syntax are not reported because present for each component. C=constraints aspect, D=dataflow aspect, G=generation aspect, TR=translation aspect, TY=typesystem aspect. Ind. = Independency, Ort. = Orthogonality, Dom. = Domain

Language component	Description	Ind.	Ort.	Size	Dom.	Completeness				
						C	D	G	TR	TY
j.m.baseLanguage	implementation of the Java language	ind.	low	-	T	✗	✗	✗	✗	
j.m.baseLanguage.logging	logging primitives	ext.	very low	-	T		✗	✗		
j.m.baseLanguage.extensionMe	extensions of classes with external methods	ext.	very low	-	T	✗		✗		✗
j.m.baseLanguage.unitTest	language to specify unit tests	ext.	low	-	T	✗	✗	✗		✗
j.m.xml	implementation of the Xml language	ind.	low	-	T	✗		✗		
j.m.xmlSchema	implementation of the Xml Schema language	ind.	low	-	T	✗		✗		
c.m.c.base	implementation of the mbeddr-C language	ind.	low	-	T	✗		✗		✗
c.m.c.pointers		ext.	very low	-	T	✗	✗	✗		✗
c.m.cc.requirements		ind.	very high	-	BO	✗		✗	✗	✗
c.m.cc.var.annotations		ext.	high	-	T	✗			✗	✗

We applied our classification to a set of language components and reported it in Table 8.2.

In particular we classified:

- components of the *baseLanguage*, which is the name used for the implemenation of Java inside MPS. The *baseLanguage* is shipped with the IDE,
- the XML and XML Schema languages, also provided by Jetbrains,
- components from mbeddr[5], an extensible variant of the C language.

8.4.2 Approach to integration

Herein we present a preliminary approach to obtain seamless language integration with full tool support in the context of language workbenches. Our reference implementation uses the Jetbrains MetaProgramming System[6] (MPS) but it is not limited to it: it could be implemented also for the Eclipse Modeling Platform [Budinsky et al., 2003] or other Language workbenches (e.g., Spoofax [Kats and Visser, 2010]) as long as the Language Workbench considered supports the languages of interest.

MPS is a projectional editor: it means that the abstract information underlying the model (something similar to the Abstract Syntax Tree) is persisted independently from the concrete syntax of the language. This is radically different from what happens with text editors. There are many benefits with this choice, but a very important one is that no parsing is necessary. In this way languages can be freely evolved and combined without the risk of obtaining an ambiguous grammar. The models are then projected, hence they are represented in a form suitable for understanding and editing by the user. Typically these projections are textual but they could also be graphical. Different artifacts can be later generated from the models: for example compiled java classes or the concrete XML files to be distributed within the compiled system.

We chose MPS for the completeness of the tool and because some of the authors acquired experience with this environment in the context of the mbeddr project[7]. Moreover MPS is distributed with language plugins which permit to to operate with Java and XML out of the box.

For the sake of simplicity and because of space constraints, we will present our approach using a working example of language interactions involving the most common pattern: references across two different languages.

---

[5]http://mbeddr.com. Note that the authors of this paper are contributors of the mbeddr project.

[6]http://www.jetbrains.com/mps/

[7]http://mbeddr.com

Let's consider a simple logging framework which obtains the configuration from an XML file. The configuration file specifies for each class the level of verbosity of the associated instance of the logger. This mechanism is for example used by Log4J[8].

We start by creating some Java classes and an XML model for the configuration of the framework. Both Java classes and the XML model are edited inside MPS.

MPS is capable, without any extension, of editing an XML model and inserting the name of the Java class we want to configure as simple text, as shown in Figure 8.1. The XML file generated is shown in Figure 8.2.

```
xml log_conf.xml

<no prolog>
<LoggingConfiguration>
  <Logger setTo="INFO">it.polito.AJavaClass</Logger>
  <Logger setTo="DEBUG">it.polito.AnotherJavaClass</Logger>
</LoggingConfiguration>
```

Figure 8.1. Editing the XML configuration file inside MPS without any extension.

```
<LoggingConfiguration>
  <Logger setTo="INFO">it.polito.AJavaClass</Logger>
  <Logger setTo="DEBUG">it.polito.AnotherJavaClass</Logger>
</LoggingConfiguration>
```

Figure 8.2. The XML file generated opned in a text editor.

This system is brittle: if the user inserts a typo, or the referred class is deleted, renamed or moved to another package the system will incur in an error which the IDE is not able to detect. Moreover the user has to type long class names (including the name of the package, to be univocal), which is both error-prone and time consuming.

Our approach consists in creating particular elements to hold references from one language inside artefacts of other languages. In this case we created a particular element which permits to represent at the appropriate semantic level a reference to a Java class inside an XML document. Using the terminology of MPS, we created a new Concept named *JavaClassRefAsXmlContent*. This concept:

- extends *XmlContent* (which represents the content of XML tags). In this way it is possible to insert instances of JavaClassRefAsXmlContent in all the places where instances of XmlContent are allowed,

[8]http://logging.apache.org/log4j/

- has a reference to the Concept *ClassConcept* (which represents Java classes). We named this reference "class",
- has specific scoping rules for the reference "class",
- when generation is invoked, has its instances substituted by the full name of the referred class. The full name is obtained concatenating the name of the package containing the class with the name of the class itself.

The implementation of this mechanism required only a few minutes.

MPS provides automatically autocompletion for the new Concept, considering the scoping rules we specified. The result obtained is visible in Figure 8.3. This mechanism in addition to offer autocompletion out of the box (therefore saving the user from writing long names, an error-prone activity), guarantees automatically consistency: renaming or moving a class the reference is automatically updated and the correct value is inserted during the generation phase. If the class is deleted the reference is recognized to be broken and the editor presents an error message, as shown in Figure 8.4. It is possible also to navigate the reference, hence from the XML file it is possible to click on the name of the referred Java class and open in in the editor.

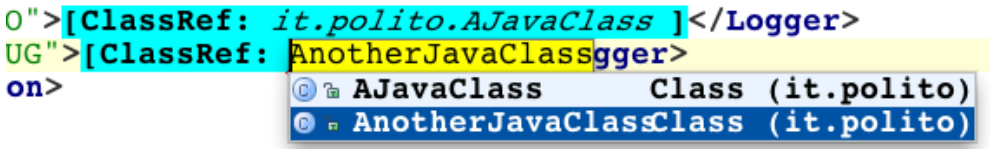

Figure 8.3. The system showing autocompletion.

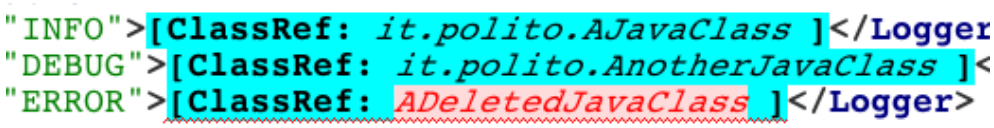

Figure 8.4. The system showing a broken reference.

This simple implementation offers a description of the level of language integration which is possible to achieve inside language workbenches with a very small effort.

While the simple example we presented deal with cross-language references, we plan to investigate also other kinds of interactions in the future.

Other mechanisms like the use of annotations of the use of custom persistence are also possible: they are not discussed in this article but they represent a future work.

8.5 Related Work

In the literature for language integration it is possible to identify two main threads: (i) approaches applicable to language families which permit to have a strong control on the definition of the languages and (ii) approaches with more general applicability.

8.5.1 Approaches Working on Family of Languages

Upon observing that the the amalgam of languages used in a single web application project are typically poorly integrated [Groenewegen and Visser, 2008], Groenewegen et al. propose the adoption of an unique language to model all the different concerns of web applications: WebDSL. They discuss the integration of an access control policy inside WebDSL. They propose to express these policies separately and then weave them inside WebDSL. In this way WebDSL remains unaware of the access control policies, while the language used to describe access control policies is created embedding knowledge of WebDSL. From this prospective we could consider this language as being part of the "family" of WebDSL. The approach of creating family of DSLs with built-in language integration is common (another example is Epsilon [Kolovos et al., 2006]). Language integration across languages of the same family, built with this particular goal in mind, is easier, w.r.t. integration among two languages not created specifically to be integrated.

Barja et al. [Barja et al., 1994] present a system based on the integration of a logic query language with an imperative programming language in the context of an object-oriented data model. They first discuss various possible approaches for the integration of the two languages and the implementation of one of them. Also in case the languages were already developed with the goal of language integration, they therefore constitute a family of DSLs.

Tolvanen et al. [Tolvanen and Kelly, 2010] describe their experience in integrating Domain Specific Modeling (DSM) languages. They do not consider integration with general purpose languages because they intend to take advantage from the fact that companies have full control of the individual DSM languages developed for internal used and how they can be integrated. This is fundamentally different from the general case of having to integrate arbitrary languages, without the possibility of modifying their definition. They discuss the case of integration based on string matching and the possibility of direct reference. They consider the possibility to combine both approaches. The second one relies on the particular technology used to realize the DSM, the MetaEdit+ system[9].

[9]MetaEdit+ Workbench 4.5 SR1 User's Guide, http://www.metacase.com/support/45/manuals/

8.5.2 General Approaches

Mayer et Schroeder [Mayer and Schroeder, 2012] name the problems of references across artifacts written in different languages as "semantic cross-language links". Being these links out of scope of the individual programming language, they are ignored by most language-specific tools and are often checked only at runtime. They propose to express explicitly constraints for these links and present three possible approaches to do that: at the source code level, using language-specific meta-models, and using language-spanning meta-models. Of these approaches they chose the second, while we advocate the third, which permits to reuse a common API and, in the case of language workbench, is already available without the necessity of developing it. Their approach is named XLL and it permits to automatically identify semantic cross-language links, correct or broken, and support the user in the activities of program understanding, analysis and refactoring. XLL requires to develop for each language considered:

- a meta-model specific for each language considered. The meta-model should contain all the information needed to individuate possible link instances, i.e. it should not necessarily represent the entire language.
- a mechanism to list the artifacts of the languages and instantiate for each of them a model pertaining to the language specific meta-model,
- an adapter to locate bindings to the refactoring capabilities of the IDE where the implementation of the approach is realized.

The approach requires then to describe possible type of links, describing for each link the nature of the element involved in the link.

Using information obtained from models of the artifacts and description of the types of links, the system is able to calculate successful and unsuccessful links. Continuous recalculation of the state of links is performed in background. The authors present a protypal implementation for the Eclipse platform and discuss two kinds of relations. The main advantage of this approach, is the possibility to be implemented as a plugin for the IDE of choice. On the other hand this approach requires a considerable effort to develop language specific adapters, and it is limited in refactoring capabilities. The authors state that their mechanism of refactoring: i) could fail under certain condition, ii) it is limited by the availability of refactoring bindings in the IDE of choice (which the authors report to be available for Java and in some form for Ruby, on the platform considered for the implementation), and iii) consider only the rename refactoring. Moreover it is not able to capture manual name changing which are not performed through explicit refactorings. Navigability seems to be limited to the artifict, not to the specific element involved in the link, which our approach permits. Both ours and their approaches support program

analysis, but while their require the development of language specific artifacts, our require almost a null effort and provide a far better tool support in respect to navigability and refactoring.

Pfeiffer realized a system called TexMo [Pfeiffer and Wasowski, 2012b] which permits to express references between artifacts written in different languages, but not to express other kind of constraints. It is realized as an Eclipse plugin and it is intended to be used instead of the original editors provided inside Eclipse. It uses a syntactic universal representation of all the languages supported. I.e., for each language has to be provided an adapter generating models (instances of the universal metamodel) from the concrete artifacts (e.g., java or xml files). Our approach do not require to recreate editors but instead permit to simply enrich the industrial-strength editors already available in MPS. It does not resoirt on a limited universal metamodel, but instead use the MPS representation of the language, which permits to consider every aspect of the language.

Pfeiffer et al. [Pfeiffer and Wasowski, 2012a] used TexMo in a controlled experiments with 22 subjects to demonstrate the effects of tool support for cross-language references. They provided to the subjects two different instances of TexMo: one with the support for cross-language references enabled and one with it disabled. Results show a significative improvemente in the ability to correctly locate the source of errors (which could be unnoticed or lead to run-time failure, when such tools support is not available). An important result is in the different way errors are located: while developers having tool-support for cross-language references locate correctly the source of errors (i.e., the broken reference), other developers barely find the effect of error, but are not able to understand the reason of the error, at least not in the short time allotted for each task (10 minutes).

8.6 Conclusions and Research Agenda

Almost every non trivial system is developed using a set of languages. While using the proper language for each task helps the productivity, the side effect is to incur in problematic language interactions which can lead to inconsistency and errors which are not recognized at development time and cause errors during the execution.

We are convinced that Language workbenches offer a solution to these problems, making possible to specify and enforce constraints. Tools can be built with a minimal effort, which provide a complete support and help developing consistent systems.

While this approach seems very promising there are different aspects which require more work. Therefore we would like to conclude this paper presenting a list of aspects which we think deserve further investigation.

Empirical assessment of the problems related to language interactions: to the best of our knowledge, we are not aware of any empirical study considering

the effects of languages interactions, with the exclusion of our previous work (see Sect. 6). The metrics provided in that preliminary study however need further validation: in breadth – more projects should be analyzed – and in depth – careful analysis of the detected interaction–. Probably better proxies to measure the level of interaction of a languages are needed. In addition to that, a more qualitative analysis on the projects already analyzed might offer both a validation of the metrics and the creation of a first catalog of the most frequent problems when languages interact. Finally, we believe it is important to keep on studying the effects on defect proneness caused by languages interactions, and to extend the investigation on the on the effect on productivity and code maintainability;

Categorize language interaction mechanisms: in the literature only cross references implemented through common identifiers are discussed. Other possible interactions are not described at all. An ontology of the kinds of interactions would be a relevant contribution. A good starting point would be the categorisation of language modularisation for DSLs as presented by Völter [Völter, 2011b].

Techniques to express cross language constraints: while the simple example we showed in section 8.4 permits to understand the potential of using Language workbenches for language integration, complete approaches have to be developed and validated in practice, possibly in different domains and on projects of different sizes;

Queries involving multiple languages: a more mature language integration would permit to query the system under development, for example about all the references to a particular element, including cross-language references. The possibilities of global queries as opposed to single-language queries have yet to be explored,

Custom persistency: the example we presented was based on the editing of XML and Java models, stored in the MPS proprietary format. One of the recent features of MPS is the possibility to use custom format for persistency (the same feature is present as well in other language workbenches). It would be possible therefore to edit XML files in MPS, while storing them as XML files, instead of using the MPS persistency format. We believe that the diverse persistency mechanisms should be investigated and assessed.

Chapter 9

Conclusions

With this thesis we had the goal to reach a better understanding on polyglot software development, considering in particular the role of modeling and domain specific languages.

Languages are the tool of the trade for software development; as the challenges faced by our field grow in complexity, the need for better tools grows as well.

Across the decades we have seen the languages we use to evolve, permitting higher level of abstractions, moving from languages shaped on the mechanics of the machines executing programs, to languages designed to fit more naturally our way of thinking.

Higher abstraction is a goal that can be pursued through different paths. One way which got some attention by practitioners in the last decades is MDD, in alternative or in combination with DSLs. We believe these approaches are particularly appealing because they empower developers with the possibility to shape their own tools and to later use them for providing solutions. We examined these approaches to better understand expectations and problems: from there we can perform a reality check useful to drive future research in higher level abstraction mechanisms.

We believe in particular in the necessity of using the best tool, hence the best language, for the task at hand. This of course leads to usage of many languages in the same project. From that a question arises: how can we blend together artifacts written in different languages in a cohesive system? Within this thesis we tried to understand the problems of language integration, and to provide some tentative solutions.

In this final chapter we first present some answers to our research questions (Sect. 9.1). Then, from the results of the first set of research questions (A.1-A.5), we derive a set of practical implications which could interest mainly practitioners (Sect. 9.2). The second set (B.1-B.4) instead contribute ideas for future research (Sect. 9.3).

We close this chapter and this thesis by providing some final remarks (Sect. 9.4).

9.1 Answers to research questions

Here we propose our answers to the research questions introduced in Sect. 1.4.

RQ A.1 *Which are the benefits expected and attained from modeling and DSL adoption?*

Modeling and DSLs are used to obtain a range of benefits which can be grouped in two clusters. The benefits of each cluster are commonly expected to be attained together. In one cluster we have *Design support*, *Maintenance support*, *Improved documentation*, in the other cluster there are *Productivity*, *Platform independence*, *Reactivity to changes*, *Quality of software*, *Flexibility* and *Standardization*. This tells us there are different ways to look at modeling.

Benefits from the first cluster are more easily attained: 52%-68% of the practitioners expecting them manage to achieve them. Benefits from the second cluster instead are rarely achieved (34%-52%).

We have seen that the most common benefits are obtained independently of the specific techniques adopted while the possibility to achieve the most difficult can be favourably affected by employing specific techniques. In this respect *Model interpretation*, *Toolsmithing* and *DSLs* can be useful.

RQ A.2 *Which are problems limiting or preventing modeling and DSL adoption?*

As for the benefits, we can also group problems in two clusters.

The first group of problems is mainly reported by practitioners who only occasionally employed modeling; these problems hinder adoption but do not completely prevent it. The other problems were instead reported by practitioners who decided to not use modeling at all; it can be seen as a set of problems preventing completely the adoption.

In particular *Refusal from developers* and *Inadequacy of supporting tools* fall in the first category, while all the other problems considered fall in the second. From the second group come the three most common problems which are: *Too much effort required*, *Not useful enough* and *Lack of competencies*.

RQ A.3 *Which are the processes and techniques used for modeling and DSL?*

UML is by far the most used modeling language, but only a few practitioners adopt UML Profiles. DSLs are rarely used. Textual DSLs are used the 50% more than graphical DSLs.

Models are written almost exclusively by technical personnel. Domain experts are involved rarely (11% of the times) and always together with technical personnel.

Among the specific techniques code generation is the most common, while model interpretation, model transformations an toolsmithing are rarely employed.

The amount of application code is commonly below 50% but there is a large variability. Automatic generation targets different parts/tiers of the system, the most common ones are SQL scripts, presentation logic and architectural code.

Model interpretation is rarer than code generation (adopters are circa one third of the adopters of code generation). It is interesting to note that companies executing models generate automatically a significantly larger amount of code than companies not executing them. This result suggests that model interpretations and code generation are not mutually exclusive alternatives. Similarly also model transformation adopters tend to generate a lot more code (around 80% of the code of each module).

These results suggest a separation between basic and advanced users, with the later group able to master and combine different sophisticated techniques.

RQ A.4 *Which are the characteristics of adoption of modeling and DSL in small companies?*

The transition to a complete model-centric development require such investments which can be often not affordable for small companies. Moreover many small companies can not shape freely their processes, because often they act as sub-contractors. For these reasons small companies tend to focus on incremental changes: they can profitably use modeling and code generation to improve the productivity of existing approaches, by complementing them rather than fully replacing them.

What seems crucial is the ability to provide benefits with a limited investment and in a limited time-frame.

On their side small companies enjoy more flexibility and willingness to experiment with new techniques. This characteristic has not to be hindered by MDD, which is often regarded as rigid.

RQ A.5 *Which are the characteristics of adoption of modeling and DSL in large companies and software development ecosystems?*

The adoption of game-changing approaches like model-centric development in large, established companies requires to face a lot of organizational challenges.

Technical problems can be reasonably easily solved with the necessary investments but a large company need to consider side aspects like model versioning, proper training, and evangelization.

Some of the problems we have seen in **RQ A.2** are particularly relevant in large companies. While small companies can easily share the knowledge about modeling, the dissemination has instead to be carefully planned and properly executed in a large company. The problem of missing competencies is particularly important because the company has to consider the availability of the relevant skills in the job market of reference.

Moreover a large company can often shape its ecosystem, instead of adapting to it, and it can afford to plan long-term investments.

RQ B.1 *How do languages interact?*

Languages interact through different mechanisms. We proposed a classification which includes:

- *Shared ID:* the same ID is used among the artifacts involved in the interaction,
- *Shared data:* a piece of data have to hold exactly the same value among the different artifacts involved,
- *Data loading:* a piece of data from one of the file involved is loaded by the code in another file involved,
- *Generation:* from one of the file involved the other files involved are completely or partially generated. Also the mod- ification of part of a file is accepted,
- *Description:* one of the file involved contained a description of the content of another file (a part or the whole file),
- *Execution:* one file execute the code contained in another file.

Of all these different kinds of interactions research focused almost exclusively on the first one, *Shared ID*. It appears to be the most common and probably also the easiest to formalize.

RQ B.2 *Which are the effects of language interactions?*

While there are different possible effects of language interactions on software development we started to explore those on defectivity. Other effects, most notably on productivity and comprehension, have still to be looked at. In the meanwhile, results from other researchers suggest that cross-language interactions greatly affect both of these aspects during maintenance activities.

Concerning defectivity we do not have unique answers: our results show that the problem is worthy to be investigated, given that most of the commits of the project examined are cross language (at least according to our coarse definition). However we also observed that this property depends on the type of the activities and the languages of the modules. Commits related to testing or fixing bugs involve mainly a single language, while adding new features or doing implementation sub-task are activities which involve multiple languages (or at least extensions). When we look at defect proneness, we observe that for Java modules the interactions with other languages (as an aggregate) is not problematic at all. However, interactions between some pairs of languages lead to a significantly higher error proneness. This is true for many relations involving C modules. Finally, the widespread interaction between Java and XML apparently is not related to defect proneness.

RQ B.3 *How can we identify language interactions?*

All the prototypical solutions proposed by other researchers are based on the manual identification and formalization of the framework specific rules determining language interactions (e.g., between Java and XML files when adopting the Spring Inversion-of-Control framework).

We investigated a fundamentally different approach which has the benefits of being general and automatic. The evaluation of our approach is still very preliminary and our results need to be confirmed with successive experiments. However we can affirm that our approach based on polyglot Abstract Syntax Trees and context exploration seems promising.

The library we developed is able to build polyglot ASTs (in which snippets of embedded languages are recognized and properly parsed). On these polyglot ASTs pairs of nodes containing the same terms are identified. These candidates are then classified as related on not depending on a comparison of their context (by context we mean the set of nodes surrounding the nodes of interest). Different metrics deriving from the Natural Languages Processing field are used to compare the contexts. Those metrics are combined using Machine learning techniques.

Very preliminary results report a precision and a recall superior to 90% for the resulting algorithm.

RQ B.4 *How can we offer tool support for language integration?*

Other researchers have realized tool support implementing specific framework support on widespread IDEs as Eclipse. Yje most relevant features are:

- *Identification of relations:* cross-language relations should be immediately visible for the developer.
- *Navigation:* the developer should be able to easily navigate between the different entities involved in the relation.
- *Refactoring support:* refactoring should preserve relations.
- *Validation:* broken relations should be identified and conveniently reported to the developer.

Differently from other approaches we believe that language workbenches are the IDEs of choice to perform language integration. Our approach, still preliminary, permits to offer all the important characteristics of required tool support with a fraction of the effort. This is possible because these tools typically implement different languages using the same underlying technology (a common meta-metamodeling facility). In this way the relations between the different languages can be easily implemented in terms of that common technology. To permit the integration between the two languages the language designer has just to introduce specific elements which acts as bridges, permitting to insert in a language sentence a reference to an element of another language. In this way the semantic gap can be easily covered permitting to offer *Identification of relations* and *Validation*. The facilities offered from the tooling permits to obtain almost automatically *Navigation* and *Refactoring support*.

9.2 Practical Implications

The adoption of modeling and DSLs is associated with the gain of a set of benefits. Unfortunately most of these benefits are not easily attained. Our first contribution to practicioners is the study of the relation between the possibility to attain a certain benefit and the adoption of specific techniques: from that we can derive guidelines to design MDD and DSLs based solutions.

The problems limiting the adoption of MDD and DSLs can be seen as technological, related to perceptions and to knowledge.

We believe technological problems are being solved by recent advances in tooling: new language workbenches permit a greater productivity. As they become

widespread we expect a reduction of practitioners reporting *Inadequacy of supporting tools* and *Too much effort required*. We as a community, need to perform a correct evangelization about the technological progresses of MDD and DSL tooling.

There is a misconception about MDD and DSLs which prevents a larger adoption (*Refusal from developers* and *Not useful enough*). From the literature we know MDD and DSLs can provide significantly benefits to adopters and our results reinforce these findings. Still, in some context there is a resistance in accepting MDD. This can be due to misconceptions developed on particular flavours of modeling (e.g., MDA and UML based solutions) or to the role changes caused by MDD. This last aspect connects to the necessity of rethinking processes and roles when moving to model-centric development.

Finally, we believe the lack of knowledge is the remaining problem hampering MDD adoption. We believe that universities need to invest more in providing the necessary skills to students, considering not only basic modeling techniques based on UML but also other topics which are currently neglected: i) UML Profiles and tools to develop DSLs, ii) model interpretation and model transformations.

9.3 Outlook

Cross-language interactions were studied by few other researchers before us. In our opinion both the directions we took working on this topic should be pursued in the future: we need to understand the phenomenon and provide solutions.

We tried to explore different aspects of the problem. At this stage we believe it is not possible to offer definitive answers, therefore we discuss an outlook to future work in all the directions we examined so far.

Forms of Cross-language relations We need to support our proposed classification with more empirical evidence. Are there other types of relations? Which are the most used ones? Quantitative data would be important to understand the relative importance of the different types of relations. It would be important also to study the typical patterns which are used with each kind of relations. While an initial list of patterns for *Shared-ID* relations was proposed [Mayer and Schroeder, 2013] we cannot confirm that list is complete moreover all other types of relations are not yet studied in depth.

Effects of Cross-language relations The effects of cross-language relations on productivity, defectivity and program comprehension have to be fully studied. Until now we are aware of our small work on defectivity (Sect. 6) and of an experiment which shows the positive effects of support for cross-language relations [Pfeiffer and Wasowski, 2012a], indirectly supporting that problems are introduced by cross-language relations.

Automatic identification of Cross-language relations Our approach is the only one we are aware of in this direction. Though it still misses a proper validation. To do that we plan to work on a golden-set to be shared with the community. This golden-set could become an important tool for comparing the performances of alternative approaches. While we already released a seed of that golden-set we plan to develop it in the future.

Tool support for Cross-language relations Tool support for specific frameworks has been implemented in some commercial or academic tools. However there are not any attempts to provide general solutions yet. In our opinion the main problem is semantic integration. This problem is greatly reduced in the context of Language Workbenches. The adoption of this kind of solutions require a significative transition cost, but the benefits for language integration are great. In such environments combining languages can be done with a very limited effort, while this is definitely not the case with traditional environment, which require a major effort to bridge the semantics of different languages. We hope to see on one side the development of solutions employing Language Workbenches of other common meta-metamodelling solutions, on the other side we think more empirical evidence is necessary, to motivate and guide successive tool implementations.

9.4 Final remarks

Recently new tools and techniques emerged which permits to craft languages, granting to programming languages an unpreceded possibility of evolving. According to Vygotsky we believe that "the relation between thought and word is a living process" [Vygotsky, 1986, p. 255], and we could iteratively revise our thoughts, as we revise the way we express them and the tools we use for the task (hence languages). Technology is providing us this ability, we have to plan how to use it in the decades to come.

In this thesis we started by studying how to foster the adoption of existing techniques for the development of single languages and later we studied how to combine those languages.

The theme of cross-language relations is still very new. Our main contribution is not to provide definitive answers but to help understand the phenomenon in its full complexity. At the same time we try also to envision solutions, which are significantly different from the ones proposed by other researchers. In particular we underline the necessity to work on general solutions for combining languages. We believe that, as the language tooling progresses, we will have more languages to be combined in the same projects. Solutions which are based on manual implementation

of support for a specific interactions risk to be not manageable.

We believe into the importance of having general solutions for language integration in the long run. Current approaches require too much specific effort to provide tool support and they are brittle, requiring continuous maintenance to support the different combinations of evolving frameworks. We envision a future in which practitioners are free to adopt and shape the pout-pourri of languages they use to develop software. To support that freedom we need solutions able to adapt to new combinations of languages, as they are created.

Bibliography

[Acerbis et al., 2007] Acerbis, R., Bongio, A., Brambilla, M., and Butti, S. (2007). WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications. In Baresi, L., Fraternali, P., and Houben, G.-J., editors, *Web Engineering*, volume 4607 of *Lecture Notes in Computer Science*, pages 501–505. Springer Berlin / Heidelberg.

[Agresti, 2007] Agresti, A. (2007). *An Introduction to Categorical Data Analysis*. Wiley-Interscience.

[Akinnaso, 1982] Akinnaso, F. N. (1982). On The Differences Between Spoken and Written Language.

[Andrews and Schneider, 1983] Andrews, G. R. and Schneider, F. B. (1983). Concepts and Notations for Concurrent Programming. *ACM Comput. Surv.*, 15(1):3–43.

[Bahli and Rivard, 2005] Bahli, B. and Rivard, S. (2005). Validating measures of information technology outsourcing risk factors. *Omega*, 33(2):175–187.

[Baker et al., 2005] Baker, P., Loh, S., and Weil, F. (2005). Model-driven engineering in a large industrial context - motorola case study. In *MoDELS '05*, pages 476–491. Springer-Verlag.

[Balasubramanian et al., 2006] Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., and Neema, S. (2006). Developing applications using model-driven design environments. *Computer*, 39(2):33–40.

[Barbosa and Alves,] Barbosa, O. and Alves, C. A Systematic Mapping Study on Software Ecosystems. In *Proceedings of the Workshop on Software Ecosystems 2011*, pages 15–26.

[Barja et al., 1994] Barja, M. L., Paton, N. W., Fernandes, A. A. A., Williams, M. H., and Dinn, A. (1994). An Effective Deductive Object-Oriented Database Through Language Integration. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 463–474, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Baroth and Hartsough, 1995] Baroth, E. and Hartsough, C. (1995). Visual object-oriented programming. chapter Visual programming in the real world, pages 21–42. Manning Publications Co., Greenwich, CT, USA.

[Bolchini et al., 2008] Bolchini, D., Garzotto, F., and Paolini, P. (2008). Investigating success factors for hypermedia development tools. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, HT '08, pages 187–192, New York, NY, USA. ACM.

[Booch, 1991] Booch, G. (1991). *Object Oriented Design With Applications*. Addison-Wesley.

[Bosch, 2009] Bosch, J. (2009). From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA. Carnegie Mellon University.

[Bosch and Bosch-Sijtsema, 2010a] Bosch, J. and Bosch-Sijtsema, P. (2010a). Coordination Between Global Agile Teams: From Process to Architecture. In v Smite, D., Moe, N. B., and AAgerfalk, P. J., editors, *Agility Across Time and Space*, pages 217–233. Springer Berlin Heidelberg.

[Bosch and Bosch-Sijtsema, 2010b] Bosch, J. and Bosch-Sijtsema, P. (2010b). From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 83(1):67–76.

[Budgen et al., 2011] Budgen, D., Burn, A. J., Brereton, O. P., Kitchenham, B. A., and Pretorius, R. (2011). Empirical evidence about the UML: a systematic literature review. *Softw. Pract. Exper.*, 41(4):363–392.

[Budinsky et al., 2003] Budinsky, F., Brodsky, S. A., and Merks, E. (2003). *Eclipse Modeling Framework*. Pearson Education.

[Carver et al., 2011] Carver, J. C., Syriani, E., and Gray, J. (2011). Assessing the Frequency of Empirical Evaluation in Software Modeling Research. In Chaudron, M., Genero, M., Abrahão, S., Mohagheghi, P., and Pareto, L., editors, *Proceedings of the First Workshop on Experiences and Empirical Studies in Software Modelling*, volume 785 of *CEUR Workshop Proceedings*, pages 20–29.

[Catal, 2009] Catal, C. (2009). Barriers to the adoption of software product line engineering. *SIGSOFT Softw. Eng. Notes*, 34(6):1–4.

[Ceri et al., 2000] Ceri, S., Fraternali, P., and Bongio, A. (2000). Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, 33(1-6):137–157. Elsevier North-Holland, Inc., New York, NY, USA.

[Clark and Chalmers, 1998] Clark, A. and Chalmers, D. J. (1998). The Extended Mind. *Analysis*, 58(1):7–19.

[Clark et al., 2004] Clark, T., Evans, A., Sammut, P., and Willans, J. (2004). *Applied Metamodelling: A Foundation for Language Driven Development (First Edition)*.

[Coplien, 1998] Coplien, J. O. (1998). *A Generative Development Process Pattern Language*. Cambridge University Press, New York.

[Cuadrado et al., 2013] Cuadrado, J. S., Izquierdo, J. L., and Molina, J. G. (2013). Applying model-driven engineering in small software enterprises. *Science of Computer Programming*, (0).

[Dahlbom, 1996] Dahlbom, B. (1996). The new informatics. *Scandinavian Journal of Information Systems*, 8(2):29–48.

[Dann et al., 2008] Dann, W. P., Cooper, S., and Pausch, R. (2008). *Learning To Program with Alice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition.

[Darwin, 1871] Darwin, C. (1871). *The descent of man, and selection in relation to sex*, volume 1. London,J. Murray,. http://www.biodiversitylibrary.org/bibliography/2092.

[Davies et al., 2006] Davies, I., Green, P., Rosemann, M., Indulska, M., and Gallo, S. (2006). How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, 58(3):358–380. Elsevier Science Publishers B. V., Amsterdam, The Netherlands.

[Day, 1988] Day, R. S. (1988). *Alternative representations*, pages 261–305. Academic Press.

[Dennett, 1996] Dennett, D. (1996). *Kinds of Minds*. BasicBooks, New York.

[Dennett, 2000] Dennett, D. C. (2000). *Making Tools for Thinking*. Oxford University Press.

[Dijkstra, 1972] Dijkstra, E. W. (1972). The humble programmer. *Commun. ACM*, 15(10):859–866.

[Dmitriev, 2004] Dmitriev, S. (2004). Language Oriented Programming: The Next Programming Paradigm. *onBoard*.

[Egorova et al., 2010] Egorova, E., Torchiano, M., and Morisio, M. (2010). Actual vs. perceived effect of software engineering practices in the Italian industry. *Journal of Systems and Software*, 83(10):1907–1916. Elsevier Science Inc., New York, NY, USA.

[Eysholdt and Behrens, 2010] Eysholdt, M. and Behrens, H. (2010). Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, SPLASH '10, pages 307–309, New York, NY, USA. ACM.

[Felleisen, 1990] Felleisen, M. (1990). On the Expressive Power of Programming Languages. In *Science of Computer Programming*, pages 134–151. Springer-Verlag.

[Fleurey et al., 2007] Fleurey, F., Breton, E., Baudry, B., Nicolas, A., and Jezequel, J. (2007). Model-Driven Engineering for Software Migration in a Large Industrial Context. In *MoDELS '07*, pages 482–497. Springer-Verlag.

[Forward et al., 2010] Forward, A., Badreddin, O., and Lethbridge, T. C. (2010). Perceptions of Software Modeling: A Survey of Software Practitioners. In *5th Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of MDD*, pages 12–24.

[Foustok, 2007] Foustok, M. (2007). Experiences in Large-Scale, Component Based, Model-Driven Software Development. In *Systems Conf., 2007 IEEE*, pages 1–8.

[Fowler and Parsons, 2011] Fowler, M. and Parsons, R. (2011). *Domain-Specific Languages*. Addison-Wesley.

[France and Rumpe, 2003] France, R. and Rumpe, B. (2003). Model engineering. *Journal Software and Systems Modeling*, 2(2):73–75. Springer-Verlag Heidelberg, Heidelberg, Germany.

[Gadamer, 1976] Gadamer, H.-G. (1976). *The Historicity of Understanding*, pages 117–133. Penguin Books Ltd.

[Gall et al., 1998] Gall, H., Hajek, K., and Jazayeri, M. (1998). Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198.

[Gamma et al., 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition.

[Gilb and Brodie, 2005] Gilb, T. and Brodie, L. (2005). Chapter 10 - Evolutionary Project Management: How to Manage Project Benefits and Costs. In *Competitive Engineering*, pages 291–319. Butterworth-Heinemann, Oxford.

[Graham, 2001] Graham, P. (2001). Beating the averages.

[Graham, 2002] Graham, P. (2002). Revenge of the nerds.

[Green, 1989] Green, T. R. G. (1989). Cognitive dimensions of notations. In Sutcliffe, A. and Macaulay, L., editors, *People and Computers V*, pages 443–460. Cambridge University Press, Cambridge, UK.

[Groenewegen and Visser, 2008] Groenewegen, D. and Visser, E. (2008). Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns. In *Web Engineering, 2008. ICWE '08. Eighth International Conference on*, pages 175–188.

[Gross, 2009] Gross, M. D. (2009). Visual languages and visual thinking: sketch based interaction and modeling. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 7–11, New York, NY, USA. ACM.

[Groves et al., 2009] Groves, R. M., Fowler, F. J., Couper, M. P., Lepkowski, J. M., Singer, E., and Tourangeau, R. (2009). *Survey Methodology*. John Wiley and Sons.

[Gumperz and Levinson, 1996] Gumperz, J. J. and Levinson, S. C. (1996). *Rethinking linguistic relativity*. Number 17 in Studies in the Social and Cultural Foundation. Cambridge University Press, Cambridge, England.

[Gurov et al., 2007] Gurov, V., Mazin, M., Narvsky, A., and Shalyto, A. (2007). Tools for support of automata-based programming. *Programming and Computer Software*, 33:343–355. MAIK Nauka/Interperiodica distributed exclusively by Springer Science+Business Media LLC.

[Hanssen, 2012] Hanssen, G. K. (2012). A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *Journal of Systems and*

Software, 85(7):1455–1466. ¡ce:title¿Software Ecosystems¡/ce:title¿.

[Harrison et al., 1997] Harrison, D. A., Mykytyn, P. P., and Riemenschneider, C. K. (1997). Executive decisions about adoption of information technology in small business: theory and empirical tests. *Information Systems Research*, 8(2):171–195.

[Hauge, 2007] Hauge, O. (2007). Open Source Software in Software Intensive Industry - A Survey. Technical report, Norwegian University of Science and Technology Department of Computer and Information Science. http://daim.idi.ntnu.no/masteroppgaver/IME/

IDI/2007/3290/masteroppgave.pdf.

[Heijstek and Chaudron, 2009] Heijstek, W. and Chaudron, M. R. V. (2009). Empirical Investigations of Model Size, Complexity and Effort in a Large Scale, Distributed Model Driven Development Process. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '09, pages 113–120. IEEE Computer Society, Washington, DC, USA.

[Hen-Tov et al., 2009] Hen-Tov, A., Lorenz, D. H., Pinhasi, A., and Schachter, L. (2009). ModelTalk: When Everything Is a Domain-Specific Language. *Software, IEEE*, 26(4):39–46.

[Hermans et al., 2009] Hermans, F., Pinzger, M., and van Deursen, A. (2009). Domain-Specific Languages in Practice: A User Study on the Success Factors. In *MoDELS*, pages 423–437.

[Hidders et al., 2005] Hidders, J., Marrara, S., Paredaens, J., and Vercammen, R. (2005). On the expressive power of XQuery fragments. In *Database Programming Languages*, pages 154–168. Springer Berlin Heidelberg.

[Hjørland and Sejer Christensen, 2002] Hjørland, B. and Sejer Christensen, F. (2002). Work tasks and socio-cognitive relevance: a specific example. *Journal of the American Society for Information Science and Technology*, 53:960–965.

[Humm and Engelschall, 2010] Humm, B. G. and Engelschall, R. S. (2010). Language-Oriented programming via DSL stacking. ICSOFT '10.

[Hutchinson et al., 2011a] Hutchinson, J., Rouncefield, M., and Whittle, J. (2011a). Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 633–642. ACM, New York, NY, USA.

[Hutchinson et al., 2011b] Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011b). Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software engineering*, ICSE '11, pages 471–480. ACM, New York, NY, USA.

[Hutchinson, 2011] Hutchinson, J. E. (2011). *An Empirical Assessment of Model Driven Development in Industry.* PhD thesis, Lancaster University.

[Ifpug, 2012] Ifpug, editor (2012). *The IFPUG Guide to IT and Software Measurement*. CRC Press.

[Iverson, 1980] Iverson, K. E. (1980). Notation as a tool of thought. *Commun. ACM*, 23:444–465.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Modeling Language Reference Manual*. Addison-Wesley.

[Jansen et al., 2009] Jansen, S., Finkelstein, A., and Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 187–190.

[Jaspars et al., 1983] Jaspars, J. M. F., Fincham, F. D., and Hewstone, M. (1983). *Attribution Theory and Research: Conceptual, Developmental, and Social Dimensions*. European Monographs in Social Psychology. Academic Press.

[Jelitshka et al., 2007] Jelitshka, A., Ciolkowski, M., Denger, C., Freimut, B., and Schlichting, A. (2007). Relevant Information Sources for Successful Technology Transfer: a Survey using Inspections as an Example. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 31–40. IEEE.

[Jiang and Hu, 2008] Jiang, D. and Hu, J. (2008). Research of Model-Based Code Automatic Generation of Management Systems. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4.

[Kasurinen et al., 2013] Kasurinen, J., Strandén, J.-P., and Smolander, K. (2013). What do game developers expect from development and design tools? In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13, pages 36–41, New York, NY, USA. ACM.

[Kats and Visser, 2010] Kats, L. C. L. and Visser, E. (2010). The Spoofax language workbench. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, SPLASH '10, pages 237–238, New York, NY, USA. ACM.

[Kelly and Tolvanen, 2008] Kelly, S. and Tolvanen, J.-P. (2008). *Domain-specific modeling: enabling full code generation*. Wiley. com.

[Kilamo et al., 2012] Kilamo, T., Hammouda, I., Mikkonen, T., and Aaltonen, T. (2012). From proprietary to open source—Growing an open source ecosystem. *Journal of Systems and Software*, 85(7):1467–1478.

[Kitchenham and Charters, 2007] Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering.

[Kitchenham and Pfleeger, 2008] Kitchenham, B. and Pfleeger, S. L. (2008). Personal Opinion Surveys. In Shull, F. and Singer, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London.

[Klein and Myers, 1999] Klein, H. K. and Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretiv Filed Studies in Information Systems. *MIS Quarterly*, 23(1):67–94.

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J., and Al, E. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc.

[Knuth, 1984] Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27:97–111.

[Kolovos et al., 2006] Kolovos, D. S., Paige, R. F., and Polack, F. A. C. (2006). The epsilon object language (EOL). In *Proceedings of the Second European conference on Model Driven Architecture: foundations and Applications*, ECMDA-FA'06, pages 128–142, Berlin, Heidelberg. Springer-Verlag.

[Leotta et al., 2012] Leotta, M., Ricca, F., Ribaudo, M., Reggio, G., Astesiano, E., and Vernazza, T. (2012). An Exploratory Survey on SOA Knowledge, Adoption and Trend in the Italian Industry. In *Proceedings of 14th International Symposium on Web Systems Evolution (WSE 2012)*, pages 21–30. IEEE.

[Lethbridge, 1998] Lethbridge, T. C. (1998). A Survey of the Relevance of Computer Science and Software Engineering Education. In *Proceedings of the 11th Conference on Software Engineering Education and Training*, pages 56–66. IEEE Computer Society, Washington, DC, USA.

[Leveque et al., 2009] Leveque, T., Estublier, J., and Vega, G. (2009). Extensibility and Modularity for Model Driven Engineering Environments. In *ECBS '09*, pages 305–314. IEEE Computer Society.

[Li et al., 2008] Li, J., Conradi, R., Petter, O., Slyngstad, N., Torchiano, M., Morisio, M., and Bunse, C. (2008). A State-of-the-Practice Survey on Risk Management in Development with Off-The-Shelf Software Components. *IEEE Software*, 34(2):271–286.

[Libkin, 2001] Libkin, L. (2001). Expressive Power of SQL.

[Logan, 2007] Logan, R. K. (2007). *The Extended Mind: The Emergence of Language, the Human Mind and Culture*. University of Toronto Press.

[Lungu et al., 2010] Lungu, M., Lanza, M., Girba, T., and Robbes, R. (2010). The Small Project Observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4):264–275.

[MacDonald et al., 2005] MacDonald, A., Russell, D., and Atchison, B. (2005). Model-driven development within a legacy system: an industry experience report. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 14–22.

[Maloney et al., 2010] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ.*, 10(4).

[Manikas and Hansen, 2013] Manikas, K. and Hansen, K. M. (2013). Software ecosystems – A systematic literature review. *Journal of Systems and Software*, 86(5):1294–1306.

[Mattsson, 2008] Mattsson, A. (2008). Automatic architectural enforcement. In Bermejo, J., Lundell, B., and van der Linden, F., editors, *Combining the Advantages of Product Lines and Open Source*, number 08142 in Dagstuhl Seminar Proceedings, pages 1–9, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[Mattsson, 2010] Mattsson, A. (2010). Automatic Enforcement of Architectural Design Rules. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 369–372, New York, NY, USA. ACM.

[Mayer and Schroeder, 2012] Mayer, P. and Schroeder, A. (2012). Cross-Language Code Analysis and Refactoring. In *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on*, pages 94–103.

[Mayer and Schroeder, 2013] Mayer, P. and Schroeder, A. (2013). Patterns of cross-language linking in java frameworks. In *21st International Conference on Program Comprehension (ICPC'13)*.

[Mellor and Balcer, 2002] Mellor, S. and Balcer, M. (2002). *Executable UML: A foundation for model-driven architecture*. Addison-Wesley.

[Mellor et al., 2003] Mellor, S. J., Clark, A. N., and Futagami, T. (2003). Model-driven development - Guest editor's introduction. *Software, IEEE*, 20(5):14–18.

[Mens et al., 2005] Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., and Jazayeri, M. (2005). Challenges in software evolution. In *Principles of Software Evolution, Eighth International Workshop on*, pages 13–22.

[Merkle, 2010] Merkle, B. (2010). Textual modeling tools: overview and comparison of language workbenches. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, SPLASH '10, pages 139–148, New York, NY, USA. ACM.

[Mitchell, 1993] Mitchell, J. C. (1993). On abstraction and the expressive power of programming languages. *Science of Computer Programming*, 21(2):141–163.

[Mohagheghi and Dehlen, 2010] Mohagheghi, P. and Dehlen, V. (2010). Where is the proof? - a review of experiences from applying MDE in industry. In *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 432–443. Springer Berlin / Heidelberg.

[Mohagheghi et al., 2009] Mohagheghi, P., Fernandez, M. A., Martell, J. A., Fritzsche, M., and Gilani, W. (2009). MDE adoption in industry : Challenges and success criteria. *Models in Software Engineering*, pages 54–59.

[Mohagheghi et al., 2012] Mohagheghi, P., Gilani, W., Stefanescu, A., and Fernandez, M. (2012). An empirical study of the state of the practice and acceptance of

model-driven engineering in four industrial cases. *Empirical Software Engineering*, pages 1–28.

[Motulsky, 2010] Motulsky, H. (2010). *Intuitive biostatistics: a nonmathematical guide to statistical thinking*. Oxford University Press.

[Navarro, 2001] Navarro, G. (2001). A Guided Tour to Approximate String Matching. *ACM Computing Survey*, 33(1):31–88.

[Navarro-Prieto and Cañas, 2001] Navarro-Prieto, R. and Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *Int. J. Hum.-Comput. Stud.*, 54(6):799–829.

[Neary and Woodward, 2002] Neary, D. S. and Woodward, M. R. (2002). An Experiment to Compare the Comprehensibility of Textual and Visual Forms of Algebraic Specifications. *J. Vis. Lang. Comput.*, 13(2):149–175.

[Nersessian, 2009] Nersessian, N. J. (2009). How Do Engineering Scientists Think? Model-Based Simulation in Biomedical Engineering Research Laboratories. *Topics in Cognitive Science*, 1.

[Nie and Zhang, 2011] Nie, K. and Zhang, L. (2011). On the Relationship between Preprocessor-Based Software Variability and Software Defects. In *High-Assurance Systems Engineering (HASE), 13th Int. Symp. on*, pages 178–179.

[Nugroho and Chaudron, 2008] Nugroho, A. and Chaudron, M. R. V. (2008). A survey into the rigor of UML use and its perceived impact on quality and productivity. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 90–99. ACM, New York, NY, USA.

[Object Management Group, 2011a] Object Management Group (2011a). Business Process Model and Notation, v. 2.0. Standard, Object Management Group.

[Object Management Group, 2011b] Object Management Group (2011b). Unified Modeling Language, Superstructure, v. 2.4. Specifications, Object Management Group.

[Oppenheim, 1992] Oppenheim, A. N. (1992). *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London.

[Pane and Myers, 2000] Pane, J. F. and Myers, B. A. (2000). Improving user performance on Boolean queries. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, pages 269–270, New York, NY, USA. ACM.

[Park, 2004] Park, K. M. (2004). Factors Contributing to Korean Students' High Achievement in Mathematics. ICME '10.

[Pfeiffer and Wasowski, 2012a] Pfeiffer, R.-H. and Wasowski, A. (2012a). Cross-Language Support Mechanisms Significantly Aid Software Development. In *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*, pages 168–184. Springer Berlin Heidelberg.

[Pfeiffer and Wasowski, 2012b] Pfeiffer, R.-H. and Wasowski, A. (2012b). TexMo: a multi-language development environment. In *Proc. of the 8th European Conf.*

on Modelling Foundations and Applications, ECMFA'12, pages 178–193, Berlin, Heidelberg. Springer-Verlag.

[Pfeiffer and Wasowski, 2013] Pfeiffer, R.-H. and Wasowski, A. (2013). Tengi Interfaces for Tracing between Heterogeneous Components. In Lammel, R., Saraiva, J., and Visser, J., editors, *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 431–447. Springer Berlin Heidelberg.

[Punter et al., 2003] Punter, T., Ciolkowski, M., Freimut, B., and John, I. (2003). Conducting on-line surveys in software engineering. In *Proceedings of the International Symposium on Empirical Software Engineering, (ISESE)*, pages 80–88.

[Pütz and Verspoor, 2000] Pütz, M. and Verspoor, M. H. (2000). *Explorations in linguistic relativity*. John Benjamins Publishing Co.

[Ratzinger et al., 2005] Ratzinger, J., Fischer, M., and Gall, H. (2005). Improving evolvability through refactoring. In *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, pages 1–5, New York, NY, USA. ACM.

[Ratzinger et al., 2007] Ratzinger, J., Sigmund, T., Vorburger, P., and Gall, H. (2007). Mining Software Evolution to Predict Refactoring. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 354–363.

[Richardson and von Wangenheim, 2007] Richardson, I. and von Wangenheim, C. G. (2007). Guest Editors' Introduction: Why are Small Software Organizations Different? *IEEE Software*, 24(1):18–22.

[Rogers, 2003] Rogers, E. M. (2003). *Diffusion of Innovations*. Simon and Schuster, 5th edition edition.

[Runeson and Höst, 2009] Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164.

[Sapir, 1929] Sapir, E. (1929). The status of linguistics as a science. *Language*, 5(209).

[Schmidt, 2006] Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39:25–31.

[Seaman, 1999] Seaman, C. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 22(4):557–572.

[Seehusen and Stølen, 2011] Seehusen, F. and Stølen, K. (2011). An Evaluation of the Graphical Modeling Framework (GMF) Based on the Development of the CORAS Tool. In Cabot, J. and Visser, E., editors, *Theory and Practice of Model Transformations*, volume 6707 of *Lecture Notes in Computer Science*, pages 152–166. Springer Berlin Heidelberg.

[Selic, 2003] Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25.

[Selic et al., 1994] Selic, B., Gullekson, G., and Ward, P. T. (1994). *Real-Time Object-Oriented Modeling*. John Wiley & Sons.

[Shirtz et al., 2007] Shirtz, D., Kazakov, M., and Shaham-Gafni, Y. (2007). Adopting Model Driven Development in a Large Financial Organization. In *Model Driven Architecture- Foundations and Applications*, volume 4530 of *Lecture Notes in Computer Science*, pages 172–183. Springer.

[Simonyi et al., 2006] Simonyi, C., Christerson, M., and Clifford, S. (2006). Intentional software. In *OOPSLA*, pages 451–464.

[Singer et al., 2008] Singer, J., Storey, M. A., and Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. In Shull, F. and Singer, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer.

[Singh et al., 2002] Singh, I., Stearns, B., and Johnson, M. (2002). *Designing Enterprise Applications with the J2EE(TM) Platform (2nd Edition)*. Prentice Hall.

[Staron, 2006] Staron, M. (2006). Adopting Model Driven Software Development in Industry A Case Study at Two Companies. In *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 57–72. Springer.

[Sutcliffe et al., 1999] Sutcliffe, A., Galliers, J., and Minocha, S. (1999). Human Errors and System Requirements. *Requirements Engineering, IEEE International Conference on*, page 23.

[Swartbooi, 2010] Swartbooi, A. A. (2010). *The role of knowledge management in offshore outsourced software development*. PhD thesis, University of Stellenbosch, Department of Information Science.

[Tolvanen and Kelly, 2010] Tolvanen, J.-P. and Kelly, S. (2010). Integrating models with domain-specific modeling languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, New York, NY, USA. ACM.

[Tomassetti et al., 2014] Tomassetti, F., Rizzo, G., and Torchiano, M. (2014). Spotting Automatically Cross-Language. In *Prooceedings of the IEEE CSMR-WCRE 2014*.

[Tomassetti et al., 2013a] Tomassetti, F., Torchiano, M., and Bazzani, L. (2013a). MDD Adoption in a Small Company: Risk Management and Stakeholders' Acceptance. *Journal of Universal Computer Science*, 19(2):186–206.

[Tomassetti et al., 2012] Tomassetti, F., Torchiano, M., Tiso, A., Ricca, F., and Reggio, G. (2012). Maturity of software modelling and model driven engineering: A survey in the Italian industry. In *Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on*, pages 91–100.

[Tomassetti et al., 2013b] Tomassetti, F., Torchiano, M., and Vetro', A. (2013b). Classification of Language Interactions. In *7th International Symposium on Empirical Software Engineering and Measurement (ESEM'13)*.

[Tomassetti et al., 2013c] Tomassetti, F., Vetro', A., Torchiano, M., Völter, M., and Kolb, B. (2013c). A Model-Based Approach to Language Integration. In *Modeling in Software Engineering (MISE), 2013 ICSE Workshop on*.

[Torchiano et al., 2011a] Torchiano, M., Di Penta, M., Ricca, F., De Lucia, A., and Lanubile, F. (2011a). Migration of information systems in the Italian industry: A state of the practice survey. *Information and Software Technology*, 53:71–86.

[Torchiano and Ricca, 2013] Torchiano, M. and Ricca, F. (2013). Six reasons for rejecting an industrial survey paper. In *Conducting Empirical Studies in Industry (CESI), 2013 1st International Workshop on*, pages 21–26.

[Torchiano et al., 2011b] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2011b). Preliminary findings from a survey on the MD* state of the practice. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 372–375. IEEE.

[Torchiano et al., 2012] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2012). Benefits from modelling and MDD adoption: expectations and achievements. In *Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling*, EESSMod '12, New York, NY, USA. ACM.

[Torchiano et al., 2013] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., and Reggio, G. (2013). Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the Italian industry. *Journal of Systems and Software*, (0).

[van Angeren et al., 2011] van Angeren, J., Blijleven, V., and Jansen, S. (2011). Relationship intimacy in software ecosystems: a survey of the Dutch software industry. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '11, pages 68–75, New York, NY, USA. ACM.

[van Deursen et al., 2007] van Deursen, A., Visser, E., and Warmer, J. (2007). Model-Driven Software Evolution: A Research Agenda. Technical report, TUDelft-SERG.

[Vetro' et al., 2011] Vetro', A., Morisio, M., and Torchiano, M. (2011). An empirical validation of FindBugs issues related to defects. In *Evaluation Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*, pages 144–153.

[Vetro' et al., 2012] Vetro', A., Tomassetti, F., Torchiano, M., and Morisio, M. (2012). Language interaction and quality issues: an exploratory study. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '12, pages 319–322, New York, NY, USA. ACM.

[Vetro' et al., 2010] Vetro', A., Torchiano, M., and Morisio, M. (2010). Assessing the precision of FindBugs by mining Java projects developed at a university. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 110–113.

[Vetro' et al., 2013] Vetro', A., Zazworka, N., Shull, F., Seaman, C., and Shaw, M. (2013). Investigating Automatic Static Analysis Results to Identify Quality Problems: an Inductive Study. In *Software Engineering Workshop (SEW), 2012 35th Annual IEEE*, pages 21–31.

[Völter, 2009] Völter, M. (2009). Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology*, 8(6):79–102.

[Völter, 2011a] Völter, M. (2011a). From programming to modeling-and back again. *Software, IEEE*, 28(6):20–25.

[Völter, 2011b] Völter, M. (2011b). Language and IDE Development, Modularization and Composition with MPS. In *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2011*, LNCS. Springer.

[Völter, 2013] Völter, M. (2013). Language and IDE Modularization and Composition with MPS. In Lämmel, R., Saraiva, J. a., and Visser, J., editors, *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 383–430. Springer Berlin Heidelberg.

[Völter and Visser, 2010] Völter, M. and Visser, E. (2010). Language extension and composition with language workbenches. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–304. ACM.

[Vygotsky, 1986] Vygotsky, L. S. (1986). *Thought and Language - Revised Edition*. The MIT Press, revised edition.

[Walonick, 1997] Walonick, D. S. (1997). *Survival Statistics*. StatPac, Inc.

[Walsham, 1993] Walsham, G. (1993). *Interpreting Information Systems in Organizations*. Wiley, Chichester, UK.

[Wampler et al., 2010] Wampler, D., Clark, T., Ford, N., and Goetz, B. (2010). Multiparadigm Programming in Industry: A Discussion with Neal Ford and Brian Goetz. *IEEE Software*, 27(5):61–64.

[Ward, 1994] Ward, M. P. (1994). Language-Oriented Programming. *Software - Concepts and Tools*, 15(4):147–161.

[Wexelblat, 1981] Wexelblat, R. L., editor (1981). *History of programming languages I*. ACM, New York, NY, USA.

[Whitehead, 1911] Whitehead, A. N. (1911). *An introduction to Mathematics*. Williams & Northgate.

[Whitley, 1997] Whitley, K. N. (1997). Visual Programming Languages and the Empirical Evidence For and Against. *Journal of Visual Languages & Computing*, 8(1):109–142.

[Whorf, 1940] Whorf, B. (1940). Science and linguistics. *Review, Technology*, 42(6).

[Wohlin et al., 2000] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers.

[Yin, 2002] Yin, R. K. (2002). *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series, Vol 5*. Sage Publications, Inc, 3rd edition.

[Zeng et al., 2005] Zeng, L., Lei, H., Dikun, M., Chang, H., Bhaskaran, K., and Frank, J. (2005). Model-driven business performance management. In *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on*, pages 295–304.